DAO Office Note 96–02

Office Note Series on Global Modeling and Data Assimilation

Richard B. Rood, Head
Data Assimilation Office
Goddard Space Flight Center
Greenbelt, Maryland

Documentation of the Physical-space Statistical Analysis System (PSAS) Part I: The Conjugate Gradient Solver Version PSAS-1.00

Arlindo da Silva*
Jing Guo †

Data Assimilation Office, Goddard Laboratory for Atmospheres
* *Goddard Space Flight Center, Greenbelt, Maryland*
† *General Sciences Corporation, Laurel, Maryland*

Goddard Space Flight Center
Greenbelt, Maryland 20771
February 1996

Abstract

This document describes Version 1 of the conjugate gradient solver component of DAO's Physical-space Statistical Analysis System (PSAS). An overview of the general PSAS algorithm is presented, followed by an outline of the pre-conditioned conjugate gradient algorithm, and its implementation in PSAS. A description of the main Fortran 90 subroutines related to the conjugate gradient solver is given, with the source code listed in the Appendix.

This Office Note focuses on a particular aspect of the PSAS algorithm, namely the conjugate gradient solver. The details of the observation and forecast error covariance modeling, the strategies for parallelization and domain decomposition, data flow and user inetrface will be described in subsequent DAO Office Notes. The emphasis of this document is on software design and implementation, and not on the scientific aspects of PSAS which will be documented elsewhere. An on-line version of this document can be obtained from

`ftp://dao.gsfc.nasa.gov/pub/office_notes/on9602.ps.Z` (postscript)

`ftp://niteroi.gsfc.nasa.gov/www/on9602/on9602.html` (HTML)

Visit also the data Assimilation Office's Home Page at

`http://dao.gsfc.nasa.gov/`

Contents

1 Introduction

The central mission of the Data Assimilation Office (DAO) is to develop a state-of-the-art Data Assimilation System capable of assimilating relevant remotely-sensed data from the Earth Observing System (EOS) platforms, as well as global atmospheric data from the other observing systems. The Physical-space Statistical Analysis System (PSAS) is a component of the Goddard EOS Data Assimilation System (GEOS/DAS) which implements a global statistical interpolation algorithm in physical rather than spectral space. This analysis system is a successor to our current *Optimal Interpolation*-based system (Pfaendtner *et al.* 1995) used to produce the GEOS-1 Multiyear assimilation (Schubert *et al.* 1993, 1995a,b). An overview of PSAS and comparisons with the Optimal Interpolation System used in Version 1 of GEOS/DAS can be found in da Silva *et al.* (1995), while some computational aspects of PSAS are discussed in Guo and da Silva (1995).

The purpose of this report is to document the software implementation of the global conjugate gradient solver in PSAS. The details of the observation and forecast error covariance modeling, the strategies for parallelization and domain decomposition, data flow and user interface will be described in subsequent DAO Office Notes.

The organization of this document is as follows. In section 2 the mathematical formulation of PSAS is introduced, with a brief overview of the whole algorithm. Section 3 describes the numerical aspects of the pre-conditioned conjugate gradient algorithm adopted in PSAS. The actual pre-conditioners used in PSAS are introduced in section 4, while section 5 provides an overview of the tasks performed in each major conjugate gradient routine. The actual Fortran 90 source code along with prologues appear in the Appendix. In the *acknowledgments* we present brief historical notes on PSAS design and development at DAO.

2 Overview of PSAS

One of the main design goals of PSAS is to provide a flexible analysis system for the assimilation of several new data types available during the EOS period. In addition, PSAS must provide the framework to test advanced forecast error covariance models, such as generic anisotropic models, and to support research on approximate Kalman filtering and smoothing at DAO. In view of this, PSAS is designed with very few assumptions on the structure of the innovation covariance matrix. Although the current implementation uses a horizontal correlation model which is homogeneous and isotropic, the numerical algorithm takes no advantage of this simplification. In contrast, most current variational systems [ECMWF's 3D-VAR (Courtier *et al.* 1993), NMC's SSI (Parrish and Derber 1992)] depend heavily on this assumption for computational feasibility. Other design goals are the elimination of data selection, and a fully global analysis system which could easily handle non-conventional data types such as satellite radiances.

PSAS implements the statistical analysis equations in physical rather spectral space. The computational advantage of a spectral formulation is tied to the assumption of isotropic horizontal error correlation structures, an assumption we would like to relax in the near future. In addition, PSAS analyses are compatible with the GEOS General Circulation Model which is formulated in grid-point space.

Formulation

Although a non-linear version of PSAS is planned, we focus our discussion on the linear aspects of the algorithm. The non-linear PSAS algorithm in consideration consists of iterations based on linear PSAS solutions.

A statistical interpolation scheme attempts to obtain an *optimal* estimate of the state of the system by combining observations with a forecast model first guess. Under a requirement of optimality the analysis equation is shown to be (*e.g.*, Daley 1991)

$$w_a = w_f + K\,(w_o - Hw_f) \tag{1}$$

$$K = P^f H^T \left(HP^f H^T + R\right)^{-1} \tag{2}$$

where $w_a \in \mathbb{R}^n$ is a vector representing the analyzed field, $w_f \in \mathbb{R}^n$ denotes the model forecast first guess, and $w_o \in \mathbb{R}^p$ is the observational vector. The operator H is a generalized interpolation operator which transforms model variables into observables. The matrix $K = P^f H^T \left(HP^f H^T + R\right)^{-1}$ is the so-called *gain* or the weights of the analysis. Typically, the number of model degrees of freedom is $n \sim 10^6$ and the current observing system has $p \sim 10^5$. The analysis equations are solved approximately by our OI system: for each grid point the weights in eq. (2) are computed with a reduced number of gridpoints $p' \ll p$, and eq. (1) is used to obtain the analyzed field. This method is clearly not feasible if all observations are to be retained. The algorithm in PSAS consists of solving one $p \times p$ linear system for the quantity y

$$\left(HP^f H^T + R\right) y = w_o - Hw_f \tag{3}$$

and subsequently obtaining the analyzed state w_a from the equation

$$w_a = w_f + P^f H^T y \tag{4}$$

which is a matrix-vector multiply plus a vector addition, requiring no iterations. The intermediate vector y will be referred to as the *partially weighted innovations*. The linear system (3) is solved by a conjugate gradient algorithm which is documented in subsequent sections.

For typical correlation models the innovation matrix $M = HP^f H^T + R$ is not sparse, although entries associated with grid points over several correlation lengths are negligibly small. In order to introduce some sparseness in M and save computational effort, zeros are introduced in M for entries corresponding to observational points distant by more than 6,000 km. For computational convenience, the sphere is divided in N regions, and matrix blocks associated with regions distant by more than 6,000 km are set to zero. For the sake of consistency and numerical stability, the tail of the correlation function must be adjusted to exactly go to zero beyond a certain distance, usually 6,000 km. For information on the construction of spatially limited correlation functions see Gaspari and Cohn (1996).

Clearly, a linear system of size $10^5 \times 10^5$ can only be solved by iterative methods. The system (3) is solved by a standard pre-conditioned conjugate gradient (CG) algorithm (Golub and van Loan, 1989). First, each row of M is normalized by the innovation variance (i.e., we solve the problem with a correlation matrix instead of a covariance matrix). The system is pre-conditioned by solving another CG problem subject to observations confined within the boundaries of each one of the N regions. These smaller CG problems are in turn pre-conditioned by solving smaller block-diagonal systems which are designed to include full vertical observational profiles, as described in section 4. These block-diagonal systems are directly solved using the Linear Algebra PACKage's (LAPACK, Anderson *et al.* 1992) Cholesky solver. In the serial implementation of PSAS, the normalized matrix M is provided as an operator, and the elements of M are recomputed each CG iteration. In the parallel implementation of PSAS being developed at the Jet Propulson Laboratory (R. Ferraro, personal communication), blocks of the matrix M are pre-computed and stored in memory. Details of the serial implementation of PSAS are given in sections 3 and 5.

As a convergence criterion for the CG solver we specify that the residual must be reduced by 1 or 2 orders of magnitude. Experiments with reduction of the residual beyond 2 orders of magnitude produced differences much smaller than the expected analysis errors. This is mainly because of the filtering properties of the operator $P^f H^T$ in (4) which attenuates the small scale details in the linear system variable y.

3 Overview of the Conjugate Gradient Algorithm

This section describes the pre-conditioned conjugate gradient algorithm from a numerical point of view; the algorithm adopted is given in Table 3. The choice of pre-conditioner in PSAS is discussed in the next section, followed by a discussion of the current Fortran 90 implementation. Readers familiar with the conjugate gradient algorithm should proceed directly to section 4.

Let $M \equiv HP^fH^T + R$ be the innovation *covariance* matrix. We start by normalizing the linear system by the diagonal of M,

$$\left(D^{-1}MD^{-1}\right)(Dy) = D^{-1}\left(w^o - Hw^f\right) \tag{5}$$

or

$$\boxed{Cx = b} \tag{6}$$

where $D_{ij} = \sqrt{M_{ij}}\delta_{ij}$. In this equation C is the innovation *correlation* matrix. Following Golub and van Loan (1989, hereafter referred to as GvL) we outline the standard pre-conditioned conjugate gradient algorithm as implemented in PSAS.

We want to solve the linear system (6) where

$$b, x \in \mathbb{R}^p \tag{7}$$

$$C \in \mathbb{R}^{p\times p} \tag{8}$$

with $p \sim 10^5$ being the number of observations. Since C is *positive definite*, solving $Cx = b$ is equivalent to finding x which minimizes the functional

$$J(x) = \frac{1}{2}x^TCx - x^Tb \tag{9}$$

The general strategy is to devise an iteration which converges to the minimum of $J(x)$ as fast as possible.

3.1 General Search Directions

Consider the iteration k,

$$x_k = x_{k-1} + \alpha_k p_k \tag{10}$$

where the step size $\alpha \in I\!R$ is a scalar and $p_k \in I\!R^p$ is a vector defining a *search direction* to be determined. It is easy to show that to minimize $J(x_{k-1} + \alpha p_k)$ with respect to α, we merely set

$$\alpha = \alpha_k = \frac{p_k^T r_{k-1}}{p_k^T C p_k} \tag{11}$$

where r_k is the *residual*

$$r_k = b - C x_k \tag{12}$$

For this choice of α we can show that

$$J(x_{k-1} + \alpha_k p_k) = J(x_{k-1}) - \frac{1}{2}\left(p_k^T r_{k-1}\right)^2 p_k^T C p_k \tag{13}$$

Notice that to ensure the reduction of J we must insist on p_k not be orthogonal to r_{k-1}.

3.2 The Steepest Descent Algorithm

The gradient of $J(x) = \frac{1}{2}x^T C x - x^T b$ with respect to x is given by

$$\nabla J|_{x=x_k} = C x_k - b \equiv -r(x_k) \tag{14}$$

The *steepest descent* algorithm looks for the minimum in the direction in which $J(x_k)$ decreases most rapidly, i.e, down-gradient

$$p_k = -\left.\nabla J\right|_{x_{k-1}} = r(x_{k-1}) \tag{15}$$

GvL give an algorithm for finding the minimum of $J(x)$ by the steepest descent method which is reproduced in Table 1.

Table 1: Steepest descent search direction algorithm (Golub and van Loan, 1989)

$k = 0$; $x_0 = 0$; $r_0 = b$
while $r_k \neq 0$
  $k = k + 1$
  $q_{k-1} = C r_{k-1}$
  $\alpha_k = r_{k-1}^T r_{k-1} / r_{k-1}^T q_{k-1}$
  $x_k = x_{k-1} + \alpha_k r_{k-1}$
  $r_k = r_{k-1} - q_{k-1}\alpha_k$
end

A known drawback of this algorithm is that convergence is too slow for matrices with large condition numbers ($\kappa_2(C) = \lambda_{max}/\lambda_{min}$, where λ is the eigenvalue of C); in this case the countours of J are elongated hyperellipsoids, and we are forced to travel back and forth *across* a valley rather then *down* a valley (there is a good discussion in Press et al. 1992). The conjugate gradient algorithm addresses this deficiency of the steepest descent method.

Table 2: Conjugate gradient algorithm (Golub and van Loan, 1989)

$$
\begin{array}{l}
k = 0;\ x_0 = 0;\ r_0 = b \\
\textbf{while } r_k \neq 0 \\
\quad k = k + 1 \\
\quad \textbf{if } k = 1 \ \{\ p_1 = r_0\ \} \\
\quad \textbf{else } \{\ \beta_k = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2} \\
\qquad\quad p_k = r_{k-1} + \beta_k p_{k-1}\ \} \\
\quad q_k = C p_k \\
\quad \alpha_k = r_{k-1}^T r_{k-1} / p_k^T q_k \\
\quad x_k = x_{k-1} + \alpha_k p_k \\
\quad r_k = r_{k-1} - \alpha_k q_k \\
\textbf{end}
\end{array}
$$

3.3 Conjugate Gradients

Recall that

$$J(x_{k-1} + \alpha_k p_k) = J(x_{k-1}) - (1/2)\left(p_k^T r_{k-1}\right)^2 p_k^T C p_k$$

To avoid the problems we encountered with the *steepest descent* algorithm, we would like to make sure we always travel in a direction perpendicular to the directions already traveled. Mathematically, we would like

$$p_j^T C p_k = 0, \qquad j < k \tag{16}$$

and, of course, we must have $p_k^T r_{k-1} \neq 0$ to ensure that J decreases in each iteration (see eq. 13). The following choice has this property

$$p_k = r_{k-1} - \frac{p_{k-1}^T C r_{k-1}}{p_{k-1}^T C p_{k-1}} p_{k-1} \tag{17}$$

It can be shown that

$$J(x_k) = \min\{J(x) | x \in \mathrm{span}\{p_1, \cdots, p_k\}\} \tag{18}$$

which guarantees global convergence and finite termination. Using a few identities (see GvL) we arrive at the algorithm given in Table 2.

Pre-conditioned Conjugate Gradients

The conjugate gradient converges as follows

$$||x - x_k||_C \leq 2||x - x_0||_C \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \tag{19}$$

where $||x||_C^2 = x^T C x$, and $\kappa = \lambda_{max}/\lambda_{min}$ is the condition number. So, convergence can be slow for large condition numbers[1]. In order to improve convergence we seek a transformation

[1] In practice, the early convergence rate depends on an *effective* condition number which is related to the *smoothness* of the RHS.

Table 3: The pre-conditioned conjugate algorithm as implemented in PSAS (Golub and van Loan, 1989)

$k = 0$; $x_0 = 0$; $r_0 = b$
while $r_k \neq 0$
  solve $\hat{C} z_k = r_k$! $\hat{C} = A^2$: preconditioner matrix
  $k = k + 1$
  if $k = 1$ $\{p_1 = z_0\}$
  else $\{\ \beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
    $p_k = z_{k-1} + \beta_k p_{k-1}\ \}$
  $q_k = C p_k$
  $\alpha_k = z_{k-1}^T r_{k-1} / p_k^T q_k$
  $x_k = x_{k-1} + \alpha_k p_k$
  $r_k = r_{k-1} - \alpha_k q_k$
end

of the original matrix C of the form,

$$\overline{C} \equiv A^{-1} C A^{-1} \tag{20}$$

where the matrix A is to be determined. Rather than solving $Cx = b$ we solve

$$\left(A^{-1} C A^{-1}\right) A x = A^{-1} b \quad \text{or} \quad \overline{C}\,\overline{x} = \overline{b} \tag{21}$$

If $A^2 \sim C$ then $\overline{C} \sim I$, and the conjugate gradient converges very fast because $\kappa(\overline{C}) \sim 1$. However, $\hat{C} \equiv A^2$ must be *simple* enough for the algorithm to be cost-effective. Usually the pre-conditioner is obtained by solving a simplified version of the problem. The pre-conditioned conjugate gradient algoritm implemented in PSAS is given in Table 3. The pre-conditioner amounts to solve an extra linear system $A^2 z_k = r_k$ every iteration. Notice that the major cost of each iteration is the matrix vector multiply operation Cp_k. Therefore, the flop counts for this algorithm scales as $\sim p^2$, *i.e.*, it scales as the square of the number of observations.

The choice pre-conditioners implemented in PSAS is discussed in the next section.

4 Choice of pre-conditioner in PSAS

The first step consists of dividing the globe into N non-overlaping geographic regions, and sorting the observations by region and data-type. For the Cray C-90 implementation we divide the globe in 80 equal-area regions using a *icosahedral* grid (Pfaendtner 1996). In the Massive Parallel implementation of PSAS being developed at JPL the globe is divided in 256 or 512 geographically irregular regions, each having approximately the same number of observations. This strategy is necessary to achieve load balance. The domain decomposition in PSAS is user specified and the different options will be documented elsewhere.

A good pre-conditioner must have two important characteristics: 1) it must be cheap to compute, and 2) it must retain the essentials of the original problem if it is to effectively improve the convergence rate of the algorithm. In fact, when we normalized the original problem by the innovation standard deviations, we indeed performed an implicit pre-conditioning. In this case the pre-conditioner approximates the original matrix by its diagonal.

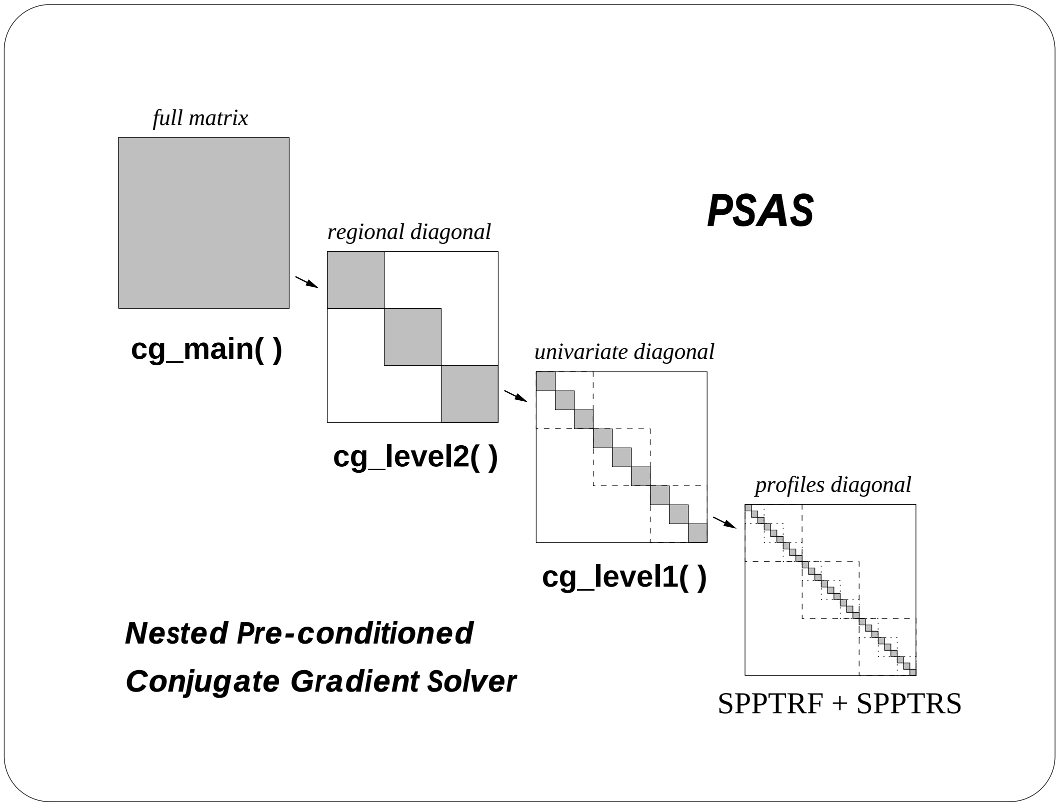

Figure 1: PSAS nested pre-conditioned conjugate gradient solver. Routine `cg_main()` contains the main conjugate gradient driver. This routine is pre-conditioned by `cg_level2()`, which solves a similar problem for each region. This routine is in turn pre-conditioned by `cg_level1()` which solves the linear system univariately. See text for details.

For the statistical interpolation problem that PSAS implements, a natural candidate for pre-conditioner is an OI-like approximation, in which the problem is solved separately for each of the N regions we used to partition the data. With $p \sim 100,000$ observations and $N \sim 80$ regions, each os these regional problems would have on average more than 1,000 observations, still too many observations for an efficient pre-conditioner. These regional problems are also solved by a pre-conditioned conjugate gradient (CG) algorithm; internally we refer to this solver as the *CG level 2*. As a pre-conditioner for *CG level 2* we solve the same problem univariately for each data type, *i. e.*, observations of u-wind, v-wind, geopotential height, etc., are treated in isolation. However, these univariate problems are still too large to be efficiently solved by direct methods and another iterative solver is used; this is the *CG level 1* algorithm. As a pre-conditioner for CG level 1 we use LAPACK (Anderson *et al.* 1992) to perform a direct Cholesky factorization of diagonal blocks of the *level 1* correlation sub-matrix. These diagonal blocks are typically of size 32, and are carefully chosen to include full vertical profiles, a desirable feature for the implementation of new data types. These nested pre-conditioned conjugate gradient solvers are illustrated in Figure 1.

5 Fortran 90 implementation of the PSAS Conjugate Gradient Solver

In this section we discuss the main Fortran 90 drivers implementing PSAS's nested conjugate gradient solver. Intentionally, we will not discuss the details of the covariance matrix-vector multiply, i.e., the step $q_k = Cp_k$ in the algorithm shown in Table 3; this complex aspect of the PSAS algorithm will be documented in a separate Office Note. A block diagram of the

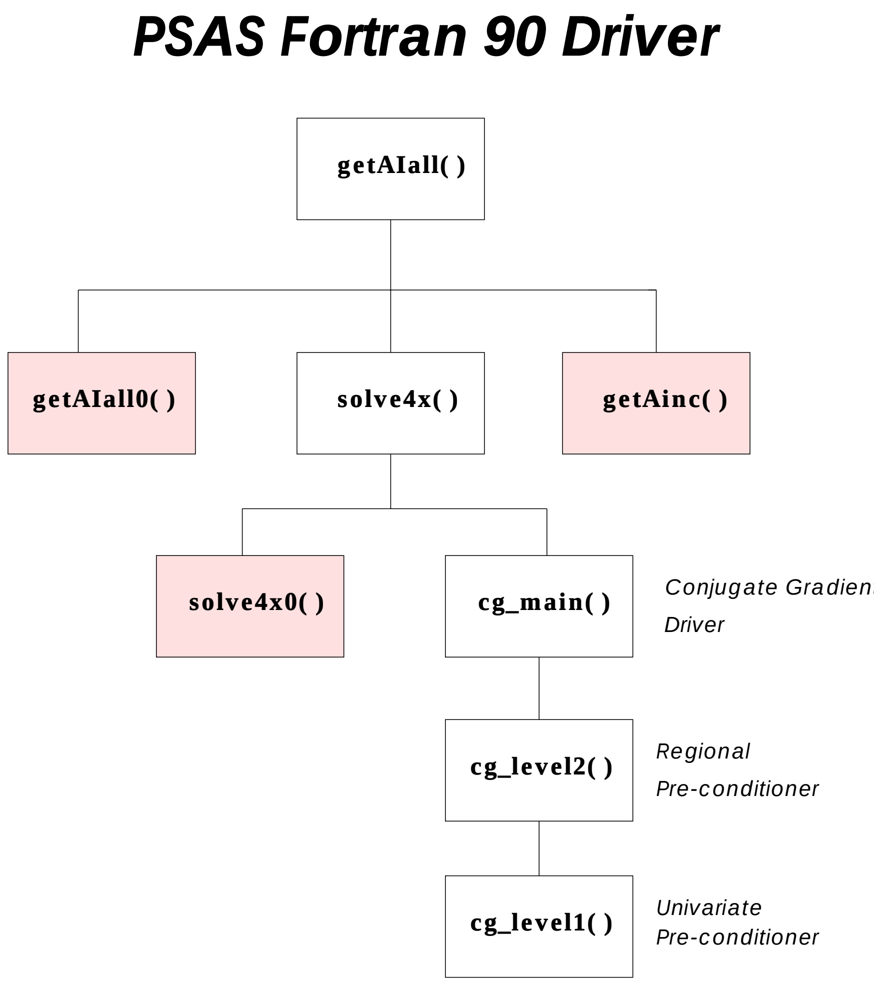

Figure 2: Block diagram of the higher level PSAS modules. The shaded blocks are not discussed in detail in this document.

modules discussed in this document is given in Figure 2; source code listing and prologue appear in the Appendix. The shaded blocks in Figure 2 are shown only for completeness; their description requires details of the covariance modeling sub-system which we do not discuss in this document.

As our starting point, we will assume that quality-controlled innovations are available, and we will discuss how the partially weighted innovations y (eq. 3) are computed. The actual calculation of the analysis increments requires the matrix-vector multiply $P^f H^T y$ (eq. 4) which cannot be discussed without going into the details of the error covariance modeling. For this reason, the module `getAinc()` shown in Figure 2 will be discussed in a separate

Office Note.

5.1 The main PSAS driver: getAIall()

This current version of this routine starts by perfoming a number of pre-processing tasks which eventually should be moved to the data ingestion section of GEOS/DAS; we have isolated this code segment inside the internal routine `psas0()`. Among these tasks are the partition of observations into regions and sorting (routine `sort()`). The following keys are used in the sorting of data

- region index
- data type index (kt)
- data source index (kx)
- latitude
- longitude
- level

After sorting, the first segment of the observation vectors will have data for region 1, then region 2, up to region N (although some regions may be empty). Inside each region all data with $kt = 1$ will be grouped together, then $kt = 2$, and so on. This sorting of the data is dictated by the strategy used for pre-conditioning described in the previous section. All routines below this point assume this sorting of the data.

Note: The current PSAS interface to GEOS/DAS is based on a customization of routine `getAIall` which processes observations and produces analysis increments in 3 separate batches, namely

surface: sea level pressure and surface winds, routine `getAIpuv()`

upper-air wind/mass: geopotential height and winds, routine `getAIzuv()`

upper-air moisture: mixing ratio, routine `getAImix()`

Because the focus of this document is on the conjugate gradient solver, we have chosen to start the PSAS driver from `getAIall()`. This interface is currently only used in the stand-alone PSAS implementation, and will eventually become the preferred GEOS/DAS interface.

5.2 Getting ready for the conjugate gradient: solve4x()

The internal routine `solve4x0()` performs several initializations, including

- Computes (x, y, z) cartesian coordinates on the unity sphere corresponding to the (lat,lon) of the input observations. These cartesian coordinates are used by the covariance modeling subsystem to compute horizontal distances.

- Computes the sounding index of the observations (da Silva and Redder 1995).
- Set interpolation indices and weights.
- Normalizes observation and forecast error standard deviations (by the innovation standard deviation).

This routine also performs normalization by the innovation standard deviation to transform the system to the form $Cx = b$ which is then handled by the conjugate gradient solver `cg_main()`.

5.3 The main conjugate gradient driver: cg_main()

This routine does a straightforward implementation of the pre-conditioned conjugate gradient algorithm given in Golub and van Loan (1989) and reproduced in Table 3; even variable names have been chosen to closely follow the book notation (with the exception perhaps, of the matrix name which we use C instead of A). The *Basic Linear Algebra Subprograms* (BLAS), which are often hand-coded in assembler and provided by several vendors, are used to perfom the basic linear algebra operations such as dot products, norms, vector additions, etc. The pre-conditioner for this routine is implemented in routine `cg_level2()`. The most costly portion of this routine is the global correlation matrix-vector multiply (routine `sCxpy`) which will be documented in a separate Office Note.

5.4 Pre-conditioner level 2: cg_level2()

This routine has a structure very similar to `cg_main()`. The main difference is how the pre-conditioner is invoked. Recall that as a result of the data sorting, within each region the observations are sorted by data-type (*e.g.*, sea level pressure, heights, u-wind, etc. are all grouped together). The pre-conditioner for this routine is implemented in routine `cg_level1()` which acts on each of these (univariate) data-type vector segments independently. In order to achieve multi-tasking on the Cray C90, this routine includes compiler directives to perform pre-conditioner operations for each data-type segments in parallel.

5.5 Pre-conditioner level 1: cg_level1()

The general structure of this routine is again similar to `cg_main()`. However, at this level the correlation block sub-matrices are explicitly computed and stored (see internal routine `cg_blocks()`. The pre-conditioner is now implemented in `cg_level0()`. This internal routine indentifies blocks of the correlation sub-matrix which contain full vertical profiles. The number of profiles is user specified; typical values are 2 or 3. A direct Cholesky solver is performed on these blocks using LAPACK (Anderson *et al.* 1992). This Cholesky solver is typically performed on matrix of size 32×32.

6 Concluding remarks

As of this writing the PSAS system is undergoing major revisions in its fundamental modules. In particular, the error covariance modeling sub-system is being updated to allow more general models (for example, non-homogeneous, non-separable correlation models), and an infra-structure for dealing with complex data-types (*e.g.*, radiances, total precipitable water) is being developed. In this document we have concentrated on the conjugate gradient solver component of PSAS. Although some revisions in these modules will be necessary as we expand some of the data structures, they will almost certainly only involve interface changes. The general structure of the algorithm appears robust and is not expected to change.

Acknowledgments

The original proposal for a global, physical-space statistical analysis system to replace DAO's OI was made by S. Cohn (1991, manuscript notes). A Fortran 77 version of PSAS was designed and implemented by the late Jim Pfaendtner during 1992–93 on his workstation. Jim Searl implementate a preliminary (univariate) version of the error covariance routines. Meta Sienkiewicz wrote the original wind-mass covariance routines and implemented the moisture analysis. David Lamich wrote the main interface to PSAS on the GEOS/DAS end (internally referred to as the "plug-version"). We would like to acknowledge their contribution and consistent encouragement during the course of this project. Thanks also to Ricky Rood (head of DAO) for overall support, and to Jim Stobie for his continued encouragement of our documentation efforts.

A formal technical review of this document was conducted on February 26, 1996 at the Data Assimilation Office. We would like to thank Meta Sienkiewicz (review leader), Genia Brin (recorder), David Lamich and Peter Lyster (reviewers) for valuable suggestions. Thanks also to Ricardo Todling for proofreading the manuscript.

A Appendix: PSAS Conjugate Gradient Solver prologues and source code

A.1 getAIall()

Given innovation (observation minus forecast) data, this routine returns the analysis increments (analysis minus first guess) using the Global conjugate gradient algorithm implemented in PSAS. Basically, the calculation is performed in 2 stages. First, a global, pre-conditioned conjugate gradient solver is used to solve for y in the equation

$$(HP^fH^T + R)y = w^o - Hw^f$$

where $w^o - Hw^f$ is the innovation. Notice that y is defined in observation locations. Subsequently, the gridded analysis increments δw_a are computed from y by the matrix-vector multiply

$$\delta w_a = P^f H^T y$$

CALLING SEQUENCE:

```
          call getAIall ( nobs, lat, lon, pres,
     &                    time, kx, kt, dels,
     &                    sig_F, sig_O,
     &                    im, jnp, mlev, pres_lev,
     &                    psl_sigF, usl_sigF, vsl_sigF,
     &                    z_sigF, u_sigF, v_sigF, mix_sigF,
     &                    psl_inc, usl_inc, vsl_inc,
     &                    z_inc, u_inc, v_inc, mix_inc,
     &                    psl_sigA, usl_sigA, vsl_sigA,
     &                    z_sigA, u_sigA, v_sigA, mix_sigA  )
```

INPUT PARAMETERS:

```
      use OEclass_tbl, only : nlev_oe, plev_oe

      implicit NONE

      integer       nobs             ! number of observations
      real          lat(nobs)        ! latitude (deg) of each obs
      real          lon(nobs)        ! longitude (deg) of each obs
      real          pres(nobs)       ! pressure level (hPa) of obs
      real          time(nobs)       ! time (minutes) from central
                                     !  synoptic time
      integer       kx(nobs)         ! GEOS/DAS data source index
```

```
integer       kt(nobs)        ! GEOS/DAS data type index
real          dels(nobs)      ! innovations (O-F)
real          sig_F(nobs)     ! forecast error stdv
real          sig_O(nobs)     ! observation error stdv (no longer
                              ! used (t. b. r.)

! -----------------------------------------------------------
! NOTE:  nobs, kx, kt, dels, sig_F & sig_O are updated during
!        the super-obing (routine proxel() ).
! -----------------------------------------------------------


integer       im               ! no. of zonal grid-points
integer       jnp              ! no. of meridional gridpoints
integer       mlev             ! no. of vertical grid-points
real          pres_lev(mlev)   ! list of vertical levels (hPa)

                               ! The arrays below with suffix
                               ! _sigF are gridded forecast error
                               ! standard deviations for:
real    psl_sigF(im,jnp)       ! o sea level pressure (hPa)
real    usl_sigF(im,jnp)       ! o surface u-wind (m/s)
real    vsl_sigF(im,jnp)       ! o surface v-wind (m/s)
real      u_sigF(im,jnp,mlev)  ! o upper-air u-wind (m/s)
real      v_sigF(im,jnp,mlev)  ! o upper-air v-wind (m/s)
real      z_sigF(im,jnp,mlev)  ! o geopotential height (m/s)
real    mix_sigF(im,jnp,mlev)  ! o mixing ratio (g/kg)
```

OUTPUT PARAMETERS:

```
                               ! The arrays below with suffix
                               ! _inc are gridded analysis
                               ! increments for:
real    psl_inc(im,jnp)        ! o sea level pressure (hPa)
real    usl_inc(im,jnp)        ! o surface u-wind (m/s)
real    vsl_inc(im,jnp)        ! o surface v-wind (m/s)
real      u_inc(im,jnp,mlev)   ! o upper-air u-wind (m/s)
real      v_inc(im,jnp,mlev)   ! o upper-air v-wind (m/s)
real      z_inc(im,jnp,mlev)   ! o geopotential height (m/s)
real    mix_inc(im,jnp,mlev)   ! o mixing ratio (g/kg)


                               ! The arrays below with suffix
                               ! _sigA are gridded analysis error
                               ! standard deviations for:
real    psl_sigA(im,jnp)       ! o sea level pressure (hPa)
real    usl_sigA(im,jnp)       ! o surface u-wind (m/s)
```

```
real    vsl_sigA(im,jnp)        ! o surface v-wind (m/s)
real      u_sigA(im,jnp,mlev)   ! o upper-air u-wind (m/s)
real      v_sigA(im,jnp,mlev)   ! o upper-air v-wind (m/s)
real      z_sigA(im,jnp,mlev)   ! o geopotential height (m/s)
real    mix_sigA(im,jnp,mlev)   ! o mixing ratio (g/kg)
```

```
Return status:

   <none> : The subroutine may exit with a non-zero status through
            a call to PSASexit() when an error condition is
            detected. In such cases execution is aborted.
```

BUGS:

```
The super-obing alters the value of observations in
violation of the ODS standard. No known side effects,
but this should be fixed.
```

SEE ALSO:

```
solve4x()    interface to conjugate gradient routines.
stdio.h      include file defining stdandard I/O units
BLAS         basic linear algebra sub-programs
```

SYSTEM ROUTINES:

```
getenv(3f)   UNIX interface returning the value of an
             environment variable (PSASRC here).
```

FILES USED:

```
stdrc        a unit number allocated when the subroutine is in use,
```

for the input of control parameters and data tables.

REVISION HISTORY:

```
ddmm95    Lamich/Guo   Interface design.
ddmm95    Guo          Initial code.
04Jan96   da Silva     Revised prologue, major clean-up.
                       Removed IFDEFs about dynamic allocation. Code
                       now requires Fortran 90 for portability.
                       Introduced getAIall0() as internal routine.
```

SOURCE CODE:

```
      character*8 myname            ! Name of routine for error messages
      parameter  (myname='getAIall')

!     Local functionality controls
!     ----------------------------
      logical want_usl
      logical want_vsl
      logical want_psl
      logical want_u
      logical want_v
      logical want_z
      logical want_mix
      parameter(want_usl=.true.)
      parameter(want_vsl=.true.)
      parameter(want_psl=.true.)
      parameter(want_u  =.true.)
      parameter(want_v  =.true.)
      parameter(want_z  =.true.)
      parameter(want_mix=.true.)

      real sigFmiss
      parameter(sigFmiss=1.e+15)

      integer   nnobs
      integer   n,ier
      integer   nprox


!     Experiment ID and date/time: debris t.b.r.
!     ------------------------------------------
      character*9   c9date
```

```
      character*8   c8time


!     Control parameters for conjugate gradient iterations
!     ----------------------------------------------------
      include     'bands.h'


!     Control parameters for output.
!     -----------------------------
      logical      verbose
      parameter  ( verbose =   .true.)


!     Basically debris from left over from JimPf time
!     -----------------------------------------------
      integer      idelprb
      integer      idelpre
      integer      idelpri
      parameter  ( idelprb =     250 ) ! beg to print dels
      parameter  ( idelpre =   20000 ) ! end to prind dels
      parameter  ( idelpri =     250 ) ! increment to print del
      logical      prtdat1
      parameter  ( prtdat1 =  .false.)
      integer      ntwidth
      parameter  ( ntwidth =   30000 )


!     Size parameters for database
!     ----------------------------
      include     'maxreg.h'    ! maximum number of regions
      include     'kxmax.h'     ! maximum numer of data sources
      include     'ktmax.h'     ! maximum number of data types
      include     'ktwanted.h'  ! data structure defining data
                                !  types for which we produce
                                !  analysis increments.


!     Regional (domain) decomposition maps used in PSAS
!     -------------------------------------------------
      integer iregbeg(maxreg)        ! pointers to beginning of regions
      integer ireglen(maxreg)        ! the no. of obs. in each region
      integer ityplen(ktmax,maxreg)  ! sizes of type blocks


!     Storage for data items (dynamic allocation)
!     -------------------------------------------
      real         sig_Ou(nobs)      ! spatially uncorrelated portion of
                                     ! obs error stdv
      real         sig_Oc(nobs)      ! spatially correlated portion of
                                     ! obs error stdv
      real         xvec(nobs)        ! Conjugate gradient solution
                                     ! at obs location
```

```
      logical      kl(nobs)         ! debris t. b. r.


      include 'lvmax.h'       ! maximum no. of levels for internal tables
      include 'levtabl.h'     ! vertical level tables for interpolation
                              !  of correlation functions, etc.

      include    'stdio.h'    ! standard I/O units

      integer    l, i
      integer    n2grd, n3grd
      integer    stdrc
      integer    luavail, lnblnk
      external   luavail, lnblnk

      external psasrcbd         ! a blockdata unit
      include 'psasrc.h'        ! a default psasrc file name

!.........................................................................


!       Initialize PSAS, sort data, assign regions, etc...
!       ---------------------------------------------------
        call getAIall0()         ! internal routine


!
!                        ---------------------
!                        COMPUTATIONAL SECTION
!                        ---------------------
!
!            Up to this point we have done a bunch of pre-processing
!            to prepare the internal data structures (forecast and
!            observation correlation tables, etc). Next we actually
!            do some real calculations for a change.
!


!       First, solve
!
!                  ( HP^fH^T + R ) x = w^o - Hw^f
!
!       for the vector x defined in observation locations.
!       ---------------------------------------------------
        call ZEITBEG('solve4x ')
        call SOLVE4X ( maxreg, iregbeg, ireglen, ityplen,
     &                 nobs, kx, lat, lon,pres,
     &                 sig_Ou, sig_Oc, sig_F, 1,
     &                 nobs, dels, xvec )
        call ZEITEND

!
        call OBSTAT ( stdout, nobs, kx, kt, pres, xvec,
     &                nlev_oe,plev_oe,'getAIall*SolutionVector')
```

```
!       Next, obtain the gridded analysis increments from
!
!                 \delta w_a = P^f H^T x
!
!       ------------------------------------------------
        call ZEITBEG ('getAinc')
        call getAinc ( verbose, stdout, nbandcg,
     &                 nobs, iregbeg, ireglen, ityplen, xvec,
     &                 lat, lon, pres, sig_F,
     &                 im, jnp, mlev, pres_lev,
     &                 usl_inc, vsl_inc, psl_inc,
     &                 u_inc, v_inc, z_inc, mix_inc,
     &                 ktwanted(ktus),
     &                 ktwanted(ktvs),
     &                 ktwanted(ktslp),
     &                 ktwanted(ktuu),
     &                 ktwanted(ktvv),
     &                 ktwanted(ktHH),
     &                 ktwanted(ktqq),
     &                 ier)
        call ZEITEND        ! getAinc


!       Error handling
!       --------------
        if(ier.ne.0) then
          write(stderr,'(2a,i4)') myname,
     &            ': error from getAinc(), ',ier
          call PSASexit ( 2, myname )
        end if


!       Scale the normalized analysis increments returned by getAinc()
!       -----------------------------------------------------------
        if(ktwanted(ktus )) call QVMV (usl_inc,usl_inc,usl_sigF,n2grd)
        if(ktwanted(ktvs )) call QVMV (vsl_inc,vsl_inc,vsl_sigF,n2grd)
        if(ktwanted(ktslp)) call QVMV (psl_inc,psl_inc,psl_sigF,n2grd)
        if(ktwanted(ktuu )) call QVMV (u_inc,u_inc,u_sigF,n3grd)
        if(ktwanted(ktvv )) call QVMV (v_inc,v_inc,v_sigF,n3grd)
        if(ktwanted(ktHH )) call QVMV (z_inc,z_inc,z_sigF,n3grd)
        if(ktwanted(ktqq )) call QVMV (mix_inc,mix_inc,mix_sigF,n3grd)


!       Print summary (means/std/min/max) of several grids
!       -------------------------------------------------
        if(ktwanted(ktus).or.ktwanted(ktvs).or.ktwanted(ktslp)) then

          write(stdout,'(/2a)') myname,
     &            ': Analysis-Increments of Surface Variables:'
```

```
       if(ktwanted(ktus)) call LVSTAT (stdout,im,jnp,usl_inc,
     &                                 0.,'WIND','SRFC',1.e+15,'USL')
       if(ktwanted(ktvs)) call LVSTAT (stdout,im,jnp,vsl_inc,
     &                                 0.,'WIND','SRFC',1.e+15,'VSL')
      if(ktwanted(ktslp)) call LVSTAT (stdout,im,jnp,psl_inc,
     &                                 0.,'PRES','SRFC',1.e+15,'SLP')

     end if

     if(ktwanted(ktuu).or.ktwanted(ktvv).or.
     &         ktwanted(ktHH).or.ktwanted(ktqq)) then

       write(stdout,'(/2a)') myname,
     &           ': Analysis-Increments of Upper-Air Variables:'

       if(ktwanted(ktuu)) call GDSTAT (stdout,im,jnp,mlev,
     &     u_inc,pres_lev,'WIND','PRES',1.e+15,'A-Inc of UWND',1)
       if(ktwanted(ktvv)) call GDSTAT (stdout,im,jnp,mlev,
     &     v_inc,pres_lev,'WIND','PRES',1.e+15,'A-Inc of VWND',1)
       if(ktwanted(ktHH)) call GDSTAT (stdout,im,jnp,mlev,
     &     z_inc,pres_lev,'HGHT','PRES',1.e+15,'A-Inc of HGHT',1)
       if(ktwanted(ktqq)) call GDSTAT (stdout,im,jnp,mlev,
     &     mix_inc,pres_lev,'MIXR','PRES',1.e+15,'A-Inc of MIXR',1)

     end if


!       Assign sigA values here.  They are initialized to zeroes for
!       now.  The operation must be conditional since the memory may
!       not be available for some calls.
!       ---------------------------------------------------------------
        call ZEITBEG ( 'getsigA' )
        if(ktwanted(ktus )) call SSCAL (n2grd,0.,usl_sigA,1)
        if(ktwanted(ktvs )) call SSCAL (n2grd,0.,vsl_sigA,1)
        if(ktwanted(ktslp)) call SSCAL (n2grd,0.,psl_sigA,1)
        if(ktwanted(ktuu )) call SSCAL (n3grd,0.,u_sigA,1)
        if(ktwanted(ktvv )) call SSCAL (n3grd,0.,v_sigA,1)
        if(ktwanted(ktHH )) call SSCAL (n3grd,0.,z_sigA,1)
        if(ktwanted(ktqq))  call SSCAL (n3grd,0.,mix_sigA,1)
        call ZEITEND


!       All done
!       --------
        l=len(psasname)+len('*')+len(myname)+len('(): normal return')
        write(stdout,'(/80a)') ('=',i=1,l)
        write(stdout,'(5a)') psasname,'*',myname,'(): normal return'
        write(stdout,'(80a)') ('=',i=1,l)

        return


        CONTAINS
!       --------
```

A.2 getAIall0()

This INTERNAL routine initializes several aspects of PSAS, including:

- Opens resource file and initializes several tables necessary for the error covariance modeling subsystem.
- Assigns a region number to each observation and set the relevant internal pointers.
- Sorts observations by region, data-type, data-source, latitude, longitude and level.
- Performs super-obing.
- Prints out lots of informational output, if specified.

CALLING SEQUENCE:

```
            call getAIall0()
```

INPUT PARAMETERS:

```
          Explicitly none, but this routine inherits all data from
its parent getAiall().
```

OUTPUT PARAMETERS:

```
          Explicitly none, but this routine resets most of the
input parameters to getAIall().
```

BUGS:

```
Most of the complexity level of this routine is due to its
provisional nature. Eventually  most of these tasks will be moved
to the data ingestion level of the data assimilation system.
```

SEE ALSO:

```
getAIall()    parent routine.
```

FILES USED:

```
stdrc       a unit number allocated when the subroutine is in use,
            for the input of control parameters and data tables.
```

REVISION HISTORY:

```
12feb96   da Silva   Moved from main body of getAIall().
```

SOURCE CODE:

```
!       Hello, world!
!       ------------
        l=len(psasname)+len('*')+len(myname)+
     &          len('(): Version_')+lnblnk(version)

        write(stdout,'(/80a)') ('=',i=1,l)
        write(stdout,'(5a)') psasname,'*',myname,'(): Version ',version
        write(stdout,'(80a)') ('=',i=1,l)


!       Total number of 2-D and 3-D gid-points
!       --------------------------------------
        n2grd = im * jnp
        n3grd = im * jnp * mlev


!       Open resource file and initialize INPAK77
!       -----------------------------------------
        stdrc=luavail()
        call GETENV ( 'PSASRC', psasrc )        ! Unix extension
```

```
      if(psasrc.eq.' ') psasrc=def_psasrc     ! default name
      call OPNINPK (stdrc,psasrc,ier)
      l=max(1,lnblnk(psasrc))
      if(ier.ne.0) then
        write(stderr,'(4a,i4)') myname,': error from opninpk(',
   &            psasrc(1:l),'), iostat = ',ier
        call PSASexit(2,myname)
      else
        write(stdout,'(4a)') myname,': using ',psasrc(1:l),
   &                  ' for runtime parameter input'
      end if


!     Initialize observation related information
!     ------------------------------------------
      call initRSRC


!     List initialized information.  Need rewrite pardisp(), since
!     so many changes have been made.  A lot of information listed by
!     pardisp() is no longer relevent, while some thing important is
!     not even listed.
!     ----------------------------------------------------------------
      c9date='01-apr-99'   ! talking about debris...
      c8time='000000'
      call PARDISP ( STDOUT,
   &                  myname,   c9date, c8time,
   &                 nobs, kxmax,  ktmax,
   &                verbose, stdout, idelprb, idelpre, idelpri,
   &                 '*****', -99, 0, 0, ntwidth,
   &                 nbands,  msmall,
   &                 cgname,  seplim, criter,  minmax,  maxpass   )


!     Print a summary of all observations.
!     ------------------------------------
      if(verbose) call OBSSMRY ( stdout, nobs, kx, kt )


!     Reset ktwanted according to the mask for this call.
!     ----------------------------------------------------
      ktwanted(ktus )=ktwanted(ktus ).and.want_usl
      ktwanted(ktvs )=ktwanted(ktvs ).and.want_vsl
      ktwanted(ktslp)=ktwanted(ktslp).and.want_psl
      ktwanted(ktuu )=ktwanted(ktuu ).and.want_u
      ktwanted(ktvv )=ktwanted(ktvv ).and.want_v
      ktwanted(ktHH )=ktwanted(ktHH ).and.want_z
      ktwanted(ktqq )=ktwanted(ktqq ).and.want_mix


!     Print out informational summaries
!     ---------------------------------
      if(ktwanted(ktus).or.ktwanted(ktvs).or.ktwanted(ktslp)) then
        write(stdout,'(/2a)') myname,
```

```
     &               ': Sigma-F of Surface Variables:'
        if(ktwanted(ktus)) call lvstat(stdout,im,jnp,usl_sigF,
     &          0.,'WIND','SRFC',sigFmiss,'USLE')
        if(ktwanted(ktvs)) call lvstat(stdout,im,jnp,vsl_sigF,
     &          0.,'WIND','SRFC',sigFmiss,'VSLE')
        if(ktwanted(ktslp)) call lvstat(stdout,im,jnp,psl_sigF,
     &          0.,'PRES','SRFC',sigFmiss,'SLPE')
      end if

      if(ktwanted(ktuu).or.ktwanted(ktvv).or.
     &   ktwanted(ktHH).or.ktwanted(ktqq)) then

        write(stdout,'(/2a)') myname,
     &          ': Sigma-F of Upper-Air Variables:'

        if(ktwanted(ktuu)) call GDSTAT(stdout,im,jnp,mlev,
     &    u_sigF,pres_lev,'WIND','PRES',sigFmiss,'Sigma-F of UWND',1)
        if(ktwanted(ktvv)) call GDSTAT(stdout,im,jnp,mlev,
     &    v_sigF,pres_lev,'WIND','PRES',sigFmiss,'Sigma-F of VWND',1)
        if(ktwanted(ktHH)) call GDSTAT(stdout,im,jnp,mlev,
     &    z_sigF,pres_lev,'WIND','PRES',sigFmiss,'Sigma-F of HGHT',1)
        if(ktwanted(ktqq)) call GDSTAT(stdout,im,jnp,mlev,
     &    mix_sigF,pres_lev,'WIND','PRES',sigFmiss,
     &          'Sigma-F of MIXR',1)

      end if


!        Restrict observations only to those 'within' at least one of
!       'hyper-boxes', defined by lat/lon/pres/kx/kt/time.  Remove data
!        outside the 'hyper-boxes' by pushing them to the end of the list
!        and reset `nobs' to the size of the front part of the list.
!       ---------------------------------------------------------------
      call ZEITBEG ('restrict')
      call RESTRICT ( verbose, stdout, nobs, prtdat1,
     &                lat, lon, pres,kx, kt,
     &                dels, sig_O, sig_F,
     &                time,nnobs                  )
      nobs = nnobs   ! completly redefine the whole data record.
      call ZEITEND


!       Sort observations in the order of:
!
!                 region(lat,lon)-kt-kx-lat-lon-pres
!
!       Also, define pointer/size information of each region and type
!       by set arrays iregbeg, ireglen, and ityplen.
!       -----------------------------------------------------------
      call ZEITBEG ('sort')

      call SORT ( myname, verbose, stdout, nobs,
     &            lat, lon, pres,
     &            kx, kt, dels,
```

```
     &              sig_O, sig_F, time,
     &              maxreg, ktmax, iregbeg, ireglen, ityplen )

        call ZEITEND


!       Remove 'duplicates' in the observations and adjust iregbeg,
!       ireglen, and ityplen accordingly.
!       ------------------------------------------------------------
        call ZEITBEG ('dupelim')
        call DUPELIM (verbose, stdout,
     &                nobs, kx, kt, kl,
     &                lat,  lon, pres,
     &                dels, sig_O, sig_F, time,
     &                maxreg, iregbeg, ireglen, ktmax, ityplen )
        call ZEITEND


!       'Superob' observations that are within a given range.  Quit
!       searching loop if nothing to 'superob', or have looped 5 times.
!       Iregbeg, ireglen, and ityplen arrays are adjusted accordingly.
!       ---------------------------------------------------------------
        call ZEITBEG ('proxel')
        nprox=0
        n=1
        do while(n.eq.1 .or. nprox.ne.0.and.n.le.5)
            call PROXEL ( verbose, stdout,
     &                    nobs, kx, kt, kl,
     &                    lat, lon, pres,
     &                    dels, sig_O, sig_F,
     &                    time, maxreg, iregbeg, ireglen,
     &                    ktmax, ityplen, nprox )
            n=n+1
        end do
        call ZEITEND


!       Reset the levels of the surface variables to 1000.
!       This way the surface analysis will use the same error
!       characteristics at the surface and at 1000 hPa.
!       -------------------------------------------------------
        do n=1,nobs
          if( kt(n).eq.ktslp .or.
     &            kt(n).eq.ktus  .or.
     &            kt(n).eq.ktvs) then
            pres(n)=1000.
          end if
        end do


!       Set the grid parameters. There apparently a good here for it
!       to be definied only now.
!       ------------------------------------------------------------
        call GRIDXX0
```

```
!       Merge in observation levels
!       ---------------------------
        call ZEITBEG('setcors')
        call SETPLEVS ( mlev,pres_lev,nobs,pres,
     &                  MXveclev,nveclev,pveclev)
        call SET_oeCHH
        call SET_fecHH
        call SET_fecQQ
        call SEThfecW      ! naming inconsistency
        call ZEITEND


!       Create observation error stdv. NOTE: in the original
!        PSAS design the observation error standard deviation
!        came along with the data stream. Due to the increasing
!        complexity of the observation error modeling, the
!        observation error is now derived from parameters in the
!        resource file. Next we overwrite whatever came in...
!       ---------------------------------------------------------
        call INTP_sigO ( nobs, kx, kt, pres, sig_Oc, sig_Ou )



!       More informational output. This time prints a summary of the
!        observations actually used in analysis
!       -------------------------------------------------------------
        if(verbose) call OBSSMRY ( stdout, nobs, kx, kt )

        call OBSTAT ( stdout, nobs,
     &                kx,kt,pres, sig_F,
     &                nlev_oe, plev_oe,
     &           'getAIall*FcstErr*sigF' )

        call OBSTAT ( stdout, nobs,
     &                kx,kt,pres, dels,
     &                nlev_oe,plev_oe,'getAIall*InnovVector')

        call OBSTAT ( stdout, nobs,
     &                kx, kt, pres, sig_Oc,
     &                nlev_oe, plev_oe, 'getAIall*ObsErr*sigOc')

        call OBSTAT ( stdout, nobs,
     &                kx, kt, pres, sig_Ou,
     &                nlev_oe, plev_oe,'getAIall*ObsErr*sigOu')


        return
        end subroutine getAIall0
```

A.3 solve4x()

Given innovation (observation minus forecast) data, this routine returns the vector y solution of the linear system of equations

$$(HP^fH^T + R)y = w^o - Hw^f$$

where $w^o - Hw^f$ is the innovation. (The notation follows da Silva and Guo 1996, DAO Office Note 9602). Notice that y is defined at observation locations. A pre-conditioned conjugate gradient algorithm is used to solve this linear system. This routine can handle multiple RHS vectors, a feature needed for the calculation of analysis error variances by means of randomized trace estimates.

CALLING SEQUENCE:

```
      call SOLVE4X ( nkr, kr_beg, kr_len, kt_len,
   &                 nobs, kx, rlat, rlon, rlev,
   &                 sigU, sig_Oc, sig_F, nvecs,
   &                 nobs_d, rhs, Xvec )
```

INPUT PARAMETERS:

```
     implicit NONE

     include   'ktmax.h'           ! maximun no. of data types
     integer   nkr                 ! number of regions
     integer   kr_beg(nkr)         ! beginning of each region
     integer   kr_len(nkr)         ! no. of obs. in each region
     integer   kt_len(ktmax,nkr)   ! no. of obs. of a given data
                                   !  in each region

     integer   nobs                ! number of observations
     integer   kx(nobs)            ! GEOS/DAS data sources
     real      rlat(nobs)          ! latitudes (deg) of obs.
     real      rlon(nobs)          ! longitudes (deg) of obs.
     real      rlev(nobs)          ! pressure levels (hPa) of obs.
     real      sig_Ou(nobs)        ! spatially uncorrelated portion
                                   !  of obs. error stdv
     real      sig_Oc(nobs)        ! spatially correlated portion
                                   !  of obs. error stdv
     real      sig_F(nobs)         ! forecast error stdv


     integer   nvecs               ! number of RHS vectors
     integer   nobs_d              ! leading dimension of RHS vector
                                   !  as declared in calling program.
```

```
                                 !  Usually nobs_d = nobs.

 real      rhs(nobs_d,nvecs)   ! RHS vectors. For the convetional
                               !  PSAS analysis system 'rhs' will
                               !  contain the innovations (O-F).
                               !  However, multiple RHS will be
                               !  necessary for implementation
                               !  analysis error variances by
                               !  randomized trace estimates.

----
NOTE:  All input arrays indexed by 'nobs' or 'nobs_d' are assumed
----   sorted by region. Within each region, data is assumed
       sorted by data type (kt). Within each data-type, data
       is assumed sorted by latitude, longitude and finally by
       levels.
```

OUTPUT PARAMETERS:

```
real       Xvec(nobs_d,nvecs)  ! solution vectors.
```

SEE ALSO:

```
cg_main()         top level conjugate gradient routine.
```

REVISION HISTORY:

```
ddmmm93  Pfaendtner  Original code.
28may93  Searl       Modification for dynamic storage on CRAY.
07jan94  Sienkiewicz Added pass of trig lat/lon.
03oct94  da Silva    Implemented CRAY specifics with IFDEFs.
                     Eliminated calls to conjgr3 . conjgr4.
                     Input parameter 'nbandmx' is now obsolete.
04oct94  da Silva    Introduced parameter nband, and call to CONJGR.
19Jan95  Guo         Added wobs tables to pass pindx2() values to
                     ??cor1() and ??corx() routines.  One could use
```

```
                  rlevs for the same purpose to reduce the over-
                  head, since rlevs has no real purpose in this
                  subroutine and subsequent routines.
02Feb95  Guo      Changed CRAY to _UNICOS for consistency and
                  to follow the guide lines.
05Feb96  da Silva Revised prologue and major clean-up.
                  Removed IFDEFs about dynamic allocation. Code
                  now requires Fortran 90 for portability.
                  Introduced internal routine solve4x0().
```

SOURCE CODE:

```
      character*7 myname
      parameter(myname='solve4x')


!     Conjugate gradient data structure
!     ---------------------------------
      include         'bands.h'


!     Dynamic allocation
!     ------------------
      real      sig_del(nobs)     ! innovation (O-F) stdv
      real      nsig_Ou(nobs)     ! normalized sig_Ou = sig_Ou/sig_del
      real      nsig_Oc(nobs)     ! normalized sig_Oc = sig_Oc/sig_del
      real      nsig_F(nobs)      ! normalized sig_F  = sig_F /sig_del

                                  ! Cartesian coordinates (on the
                                  ! unity sphere) of unit vectors
                                  ! of the spherical coordinate system
      real      qr_x(nobs)        ! o x-coord of radial       unit vector
      real      qr_y(nobs)        ! o y-coord of radial       unit vector
      real      qr_z(nobs)        ! o z-coord of radial       unit vector
      real      qm_x(nobs)        ! o x-coord of meridional   unit vector
      real      qm_y(nobs)        ! o y-coord of meridional   unit vector
      real      qm_z(nobs)        ! o z-coord of meridional   unit vector
      real      ql_x(nobs)        ! o x-coord of longitudinal unit vector
      real      ql_y(nobs)        ! o y-coord of longitudinal unit vector
                                  ! NOTE: ql_z is not needed.

                                  ! Interpolation indices/weights:
      integer   ktab(nobs)        ! o vertical   interpolation index
      real      wtab(nobs)        ! o vertical   interpolation weights
      integer   jtab(nobs)        ! o meridional interpolation index
      real      vtab(nobs)        ! o meridional interpolation weights

      integer   ks(nobs)          ! sounding index
```

```
      real      bvec(nobs,nvecs)  ! normalized RHS = RHS / sig_del


!     Levels for correlation tables, etc.
!     -----------------------------------
      include 'lvmax.h'
      include 'levtabl.h'
      include 'hfecW.h'

      include 'stdio.h'           ! standard i/o

!     Local variables
!     ---------------
      integer  i                 ! data index
      integer  ivec              ! vector index
      integer  ierr              ! error code

      real     var

!.......................................................................


!      Compute cartesian coordinates and set interpolation indices
!      -----------------------------------------------------------
       call solve4x0()


!      Normalize the the RHS vectors
!      -----------------------------
       do ivec=1,nvecs
          do i=1,nobs
             bvec(i,ivec) = rhs(i,ivec) / sig_del(i)
          end do
       end do


!     Use conjugate gradient algorithm to solve the normalized
!      linear system based on the innovation CORRELATION matrix,
!     i.e., the CG solver works on the system
!
!                           C x = b
!
!     where C is the innovation correlation matrix and b is
!     (usually) the innovation normalized by its standard deviation
!     -------------------------------------------------------------
      call CG_MAIN ( cgverb(nbandcg),
     &               nkr, kr_beg, kr_len, kt_len,
     &               nobs, ks, nsig_Ou, nsig_Oc, nsig_F,
     &               qr_x, qr_y, qr_z,
     &               qm_x, qm_y, qm_z,
     &               ql_x, ql_y, ktab,
     &               wtab, jtab, vtab,
     &               nvecs, nobs, bvec, nobs, Xvec, ierr )
```

```
!     Error handling
!     --------------
      if ( ierr .ne. 0 ) then
          write(stderr,'(2a,i3)') myname,
     &                 ': error from cg_main(), ',ierr
          call PSASexit ( 2, myname )
      end if


!     Scale solution by the innovation standard deviation
!     ---------------------------------------------------
      do ivec = 1, nvecs
         do i = 1, nobs
            Xvec(i,ivec) = sig_del(i) * Xvec(i,ivec)
         end do
      end do


!     All done
!     --------

      return


      CONTAINS
!     --------
```

A.4 solve4x0()

This INTERNAL Fortran 90 routine initializes several internal parameters relevant to the conjugate gradient solver, including

- Computes (x, y, z) cartesian coordinates on the unity sphere corresponding to the (lat,lon) of the input observations. These cartesian coordinates are used by the covariance modeling subsystem to compute horizontal distances.
- Computes the sounding index of the observations.
- Set interpolation indices and weights.
- Normalizes observation and forecast error standard deviations (by the innovation standard deviation).

CALLING SEQUENCE:

```
          call solve4x0()
```

INPUT PARAMETERS:

```
          Explicitly none, but this routine inherits all data from
  its parent solve4x().
```

OUTPUT PARAMETERS:

```
          Explicitly none, but this routine sets several quantities
  of relevance to the conjugate gradient solver.
```

SEE ALSO:

```
       solve4x()    parent routine.
```

REVISION HISTORY:

```
12feb96   da Silva   Moved from main body of solve4x().
```

SOURCE CODE:

```
!      Compute x,y,z coordinates of observations
!      -----------------------------------------
       call LL2QVEC ( nobs,rlat,rlon,
     &                qr_x,qr_y,qr_z,qm_x,qm_y,qm_z,ql_x,ql_y)


!      Set sounding index of observations
!      ----------------------------------
       call SETPIX  ( nobs, kx, rlat, rlon, ks )


!      Set tables for vertical/horizontal interpolation
!      ------------------------------------------------
       call SLOGTAB (.true., nveclev,pveclev,nobs,rlev,ktab,wtab)
       call SLINTAB (.true., nHlat,Hlat,nobs,rlat,jtab,vtab)


!      Compute normalized error stdv
!      -----------------------------
       do i=1,nobs
          var=sig_Ou(i)*sig_Ou(i)+sig_Oc(i)*sig_Oc(i)+sig_F(i)*sig_F(i)
          sig_del(i) = 1. / sqrt(var)
          nsig_Ou(i) = sig_Ou(i) / sig_del(i)
          nsig_Oc(i) = sig_Oc(i) / sig_del(i)
          nsig_F(i)  = sig_F(i)  / sig_del(i)
       end do

      return


      end subroutine SOLVE4X0
```

A.5 cg_main()

Solves the linear system of equations

$$Cx = b$$

where C is the innovation correlation matrix, and b is a set of multiple RHS. When perfoming a global analysis with PSAS, the RHS is simply the innovation (O-F) normalized by its standard deviation. The multiple RHS are necessary to estimate analysis error variances by means of randomized trace estimates.

The *Pre-conditioned Conjugate Gradient* algorithm is standard and closely follows

Golub, G. H. and C. F. van Loan, 1989: *Matrix Computations*, 2nd Edition, The John Hopkins University Press, 642pp.

and is reproduced below.

$k = 0$; $x_0 = 0$; $r_0 = b$
while $r_k \neq 0$
  solve $\hat{C} z_k = r_k \Longrightarrow$ `call cg_level2()`
  $k = k + 1$
  if $k = 1$ $\{ p_1 = z_0 \}$
  else $\{ \beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
    $p_k = z_{k-1} + \beta_k p_{k-1} \}$
  $q_k = C p_k \qquad \Longrightarrow$ `call sCxpy()`
  $\alpha_k = z_{k-1}^T r_{k-1} / p_k^T q_k$
  $x_k = x_{k-1} + \alpha_k p_k$
  $r_k = r_{k-1} - \alpha_k q_k$
end

Notice that `cg_level2()` implements the pre-conditioner which consists of solving the same problem using only regional diagonal blocks of the the correlation matrix C.

The practical implementation below stops the iteration before exact convergence. Indeed, the iteration stops if we exceed a pre-determined maximum number of iterations or the residual is reduced by a specified number of orders of magnitude. These options are selected via the PSAS resource file (usually named `psas.rc`).

CALLING SEQUENCE:

```
      call CG_MAIN ( verbose,
     &               nkr, kr_beg, kr_len, kt_len,
     &               nobs, ks, nsig_Ou, nsig_Oc,  nsig_F,
     &               qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,
     &               ktab, wtab, jtab, vtab,
     &               nvecs, nobs_d, b, x, ierr )
```

INPUT PARAMETERS:

```
implicit  NONE

logical   verbose             ! if .true. prints out all kind
                              !  of informational output to stdout.

include   'ktmax.h'           ! maximun no. of data types
integer   nkr                 ! number of regions
integer   kr_beg(nkr)         ! beginning of each region
integer   kr_len(nkr)         ! no. of obs. in each region
integer   kt_len(ktmax,nkr)   ! no. of obs. of a given data type
                              !  in each region

integer   nobs                ! number of observations
integer   ks(nobs)            ! sounding index

                              ! Observation/forecast errors stdv
                              ! normalized by innovation (O-F) stdv:
real      nsig_Ou(nobs)       ! o normalized spatially uncorrelated
                              !   observation error stdv
real      nsig_Oc(nobs)       ! o normalized spatially correlated
                              !   observation error stdv
real      nsig_F(nobs)        ! o normalized forecast error stdv


                              ! Cartesian coordinates (on the
                              ! unity sphere) of unit vectors
                              ! of the spherical coordinate system
real      qr_x(nobs)          ! o x-coord of radial       unit vector
real      qr_y(nobs)          ! o y-coord of radial       unit vector
real      qr_z(nobs)          ! o z-coord of radial       unit vector
real      qm_x(nobs)          ! o x-coord of meridional   unit vector
real      qm_y(nobs)          ! o y-coord of meridional   unit vector
real      qm_z(nobs)          ! o z-coord of meridional   unit vector
real      ql_x(nobs)          ! o x-coord of longitudinal unit vector
real      ql_y(nobs)          ! o y-coord of longitudinal unit vector
                              ! NOTE: ql_z is not needed.


                              ! Interpolation indices/weights:
integer   ktab(nobs)          ! o vertical   interpolation index
real      wtab(nobs)          ! o vertical   interpolation weights
integer   jtab(nobs)          ! o meridional interpolation index
real      vtab(nobs)          ! o meridional interpolation weights
```

```
integer   nvecs               ! Number of RHS vectors
integer   nobs_d              ! leading dimension of RHS vector
                              !  as declared in calling program.
                              !  Usually nobs_d = nobs.

real      b(nobs_d,nvecs)     ! RHS vectors  normalized by
                              !  innovation stdv. For the convetional
                              !  PSAS analysis system 'b' will
                              !  contain the normalized innovations
                              !  (O-F). However, multiple RHS will be
                              !  necessary for implementation
                              !  analysis error variances by
                              !  randomized trace estimates.
```

OUTPUT PARAMETERS:

```
real       x(nobs_d,nvecs)    ! Solution vectors.
integer    ierr               ! error return code. All is well
                              ! if ierr=0.
```

SEE ALSO:

```
cg_level2()    Pre-conditioner routine.
stdio.h        Include file defining stdandard I/O units
BLAS           Basic linear algebra sub-programs
```

REVISION HISTORY:

```
03apr93   Pfaendtner  Original code
04jun93   Pfaendtner  Modification for dynamic storage on CRAY
07jan94   Sienkiewicz  Added pass of trig lat/lon to subroutine
14feb94   da Silva    Fixed search direction bug
09apr94   Pfaendtner  Added prologue
13apr94   Pfaendtner  Added use of libsci routines
03oct94   da Silva    Implemented CRAY specifics with IFDEFs.
04oct94   da Silva    Routine changed name from CONJGR5 to
```

```
                      simply CONJGR. Introduced parameter
                      nbandmx.
  19Jan95   Guo       Added wobs tables to pass pindx2() values to
                      ??cor1() and ??corx() routines.  One could use
                      rlevs for the same purpose to reduce the over-
                      head, since rlevs has no real purpose in this
                      subroutine and subsequent routines.
  02Feb95   Guo       Changed CRAY to _UNICOS for consistency and
                      to follow the guide lines.
  11Oct95   Guo       Summary of changes since 02Feb95:
                        + some structural changes for multitasking on
                          C90, including now handling all regions in
                          one conjgr2() call.
                        + modified to accept multi vectors;
                        + replaced multbyC() call to sCxpy() call;
  06Feb96   da Silva  Revised prologue, and several minor changes
                      for readability:
                        o name change: from conjgr() to cg_main()
                        o removed static allocation IFDEFs;
                          code now requires Fortran 90 for
                          portability.
                        o simplified main loop
                        o several variable name changes to conform
                          to notation in Golub and van Loan;
                          comments are straight quotation from book.
                        o introduction of f90 assignments whenever
                          possible.
```

SOURCE CODE:

```
        character*7  myname
        parameter   (myname='cg_main')


!       Local storage (dynamic allocation)
!       ----------------------------------
        include      'mxpass.h'          ! max dimension for sizerr
        real          x_k(nobs,nvecs)    ! solution at kth iteration
        real          r_k(nobs,nvecs)    ! residual at kth iteration
        real          z_k(nobs,nvecs)    ! pre-conditioner at kth iteration
        real          p_k(nobs,nvecs)    ! search direction at kth iteration
        real         Cp_k(nobs,nvecs)    ! Correlation matrix * p_k
        real      r_norm(0:mxpass,nvecs) ! residual norm

        real     zTr_new(nvecs)          ! z' * r   (new)
        real     zTr_old                 ! z' * r   (old)
                                         ! where z' = transpose(z)
```

```
!       Defines kind of covariance matrices
!       -----------------------------------
        integer kind_mat, kind_cov
        include 'kind_mats.h'
        include 'kind_covs.h'


!       Convergence control parameters.
!       ------------------------------
        include     'bands.h'

        include     'stdio.h'


!       BLAS functions
!       --------------
        real      sdot, snrm2
        external  sdot, snrm2


!       Minor local variables not worth commenting
!       ------------------------------------------
        real     alpha_k, beta, tol
        integer  k, kn, ivec, k_max
        integer  i

!.......................................................................


        call ZEITBEG (cgname(nbandcg))   ! starts timing


!       Initialization: k=0; x_0=0; r_0=b
!       ---------------------------------
        k = 0
        x_k = 0.
        do ivec=1,nvecs
           r_k(1:nobs,ivec) = b(1:nobs,ivec)
           r_norm(k,ivec) = SNRM2(nobs,r_k(1,ivec),1)
        end do
        kn = 0

!       Iterate...
!       ----------
        k_max = maxpass(nbandcg)
        tol = criter(nbandcg)
        DO WHILE ( k .le. k_max .and.
     &            (r_norm(kn,1)/r_norm(0,1)) .gt. tol )

           k = k + 1


!          Pre-conditioner step:  Solve \hat{C} z_k = r_k
!          ----------------------------------------------
```

```
      call CG_LEVEL2 ( verbose.and.cgverb(2), kind_cov0.or.kind_covF,
     &                 nkr, kr_beg, kr_len, kt_len,
     &                 nobs, ks, nsig_Ou, nsig_Oc, nsig_F,
     &                 qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,
     &                 ktab, wtab, jtab, vtab,
     &                 nvecs, nobs, r_k, z_k, ierr )


!     Error handling
!     --------------
      if ( ierr .ne. 0 ) then
           if ( ierr .lt . 0 ) then
                write(stderr,*) myname,
     &          ': insufficient working space in cg_level2(), ',
     &          'size = ',-ierr
           else
             write(stderr,'(3a,i6)') myname,
     &                     ': unexpected return from cg_level2(), ',
     &                     'err = ',ierr
           end if
           call PSASexit(2,myname)
      end if


!     Set search direction, p_k.
!     --------------------------
      do ivec=1,nvecs

!        if k = 1 { p_1 = z_0 }
!        ----------------------
         if( k.eq.1 ) then

             zTr_new(ivec) = SDOT(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
             call SCOPY(nobs,z_k(1,ivec),1,p_k(1,ivec),1)


!        else { beta_k = r_{k-1}^T z_{k-1} / r_{k-1}^T z_{k-2}
!               p_k = z_k-1 + \beta_k p_{k-1} }
!        ------------------------------------------------------
         else

             zTr_old = zTr_new(ivec)
             zTr_new(ivec) = SDOT(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
             beta = zTr_new(ivec) / zTr_old
             call SAXPY(nobs,beta,p_k(1,ivec),1,z_k(1,ivec),1)
             call SCOPY(nobs,z_k(1,ivec),1,p_k(1,ivec),1)

          end if

      end do      ! loop over RHS vectors


!     q_k = C p_k
!     -----------
```

```
      kind_mat=nbandcg
      call sCxpy ( kind_mat, kind_covO .or. kind_covF,
     &             nkr, kr_beg, kr_len, kt_len,
     &             nobs, ks, nsig_Ou, nsig_Oc, nsig_F,
     &             qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,
     &             ktab, wtab, jtab, vtab,
     &             nvecs, nobs, p_k, nobs, Cp_k,
     &             ierr )


!     Error handling
!     --------------
      if ( ierr .ne. 0 ) then
           if(ierr.lt.0) then
             write(stderr,'(3a,i10)') myname,
     &               ': insufficient working space in sCxpy(), ',
     &               'size = ',-ierr
           else
             write(stderr,'(3a,i3)') myname,
     &                    ': unexpected return from sCxpy(), ',
     &                    'err = ',ierr
           end if
           call PSASexit(2,myname)
      end if



!     For each RHS vector
!     -------------------
      do ivec=1,nvecs

!        alpha_k = z_{k-1}^T r_{k-1} / p_k^T q_k
!        ---------------------------------------
         alpha_k = zTr_new(ivec) /
     &             SDOT(nobs,p_k(1,ivec),1,Cp_k(1,ivec),1)

!        x_k = x_{k-1} + alpha_k p_k
!        ---------------------------
         call SAXPY(nobs, +alpha_k, p_k(1,ivec),1,x_k(1,ivec),1)

!        r_k = r_{k-1} - alpha_k q_k
!        ---------------------------
         call SAXPY(nobs,-alpha_k,Cp_k(1,ivec),1,r_k(1,ivec),1)

      end do


!     Residual norm at end of this iteration
!     --------------------------------------
      kn = kn + 1
      if ( kn .gt. MXPASS ) kn = 1   ! cyclic storage
      do ivec=1,nvecs
         r_norm(kn,ivec) = SNRM2(nobs,r_k(1,ivec),1)
      end do
```

```
      END DO  ! end of CG iteration


!     Convergence achieved
!     --------------------
      if( (r_norm(kn,1)/r_norm(0,1)) .le. tol .and. verbose ) then
          write(stdout,'(2a)') myname,': convergence achieved'


!     Maximum number of iterations exceeded
!     -------------------------------------
      else if ( verbose ) then
          write(stdout,'(2a)') myname,
     &                    ': maximum number of iterations exceeded'

      end if


!     Print summary
!     -------------
      if ( verbose ) then
          call CGNORM ( myname, criter(nbandcg), mxpass,
     &                  k, nvecs, r_norm, nobs)
      end if


!     Return kth iterate as solution
!     ------------------------------
      do ivec=1,nvecs
         x(1:nobs,ivec) = x_k(1:nobs,ivec)
      end do


!     All done
!     --------
      call ZEITEND

      return
      end
```

A.6 cg_level2()

Solves the linear system of equations

$$\tilde{C}x = b$$

where $\tilde{C}$ is a simplified version of innovation covariance matrix, and b is a set of multiple RHS. The matrix $\tilde{C}$ consists of regional diagonal blocks of the the correlation matrix C. This routine is meant to be a pre-conditioner for routine `cg_main()`. When perfoming a global analysis with PSAS, the RHS is simply the innovation (O-F) normalized by its standard deviation. The multiple RHS are necessary for the estimate of analysis error variances by means of randomized trace estimates.

The *Pre-conditioned Conjugate Gradient* algorithm is standard and closely follows

Golub, G. H. and C. F. van Loan, 1989: *Matrix Computations*, 2nd Edition, The John Hopkins University Press, 642pp.

and is reproduced in the prologue of routine `cg_main()`. The pre-conditioner for this routine is implemented in `cg_level1()`. This pre-conditoner solves a similar problem, this time univariately.

CALLING SEQUENCE:

```
      call CG_LEVEL2 ( verbose, kind_cov,
   &                   nkr, kr_beg, kr_len, kt_len,
   &                   nobs, ks, nsig_Ou, nsig_Oc,  nsig_F,
   &                   qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,
   &                   ktab, wtab, jtab, vtab,
   &                   nvecs, nobs_d, b, x, ierr )
```

INPUT PARAMETERS:

```
   logical   verbose              ! if .true. prints out all kind
                                  !  of informational output to stdout.

   integer   kind_cov             ! specifies the kind of covariance
                                  ! matrix.

   include   'ktmax.h'            ! maximun no. of data types
   integer   nkr                  ! number of regions
   integer   kr_beg(nkr)          ! beginning of each region
   integer   kr_len(nkr)          ! no. of obs. in each region
   integer   kt_len(ktmax,nkr)    ! no. of obs. of a given data type
```

```
                                ! in each region

      integer   nobs            ! number of observations
      integer   ks(nobs)        ! sounding index

                                ! Observation/forecast errors stdv
                                ! normalized by innovation (O-F) stdv:
      real      nsig_Ou(nobs)   ! o normalized spatially uncorrelated
                                !   observation error stdv
      real      nsig_Oc(nobs)   ! o normalized spatially correlated
                                !   observation error stdv
      real      nsig_F(nobs)    ! o normalized forecast error stdv


                                ! Cartesian coordinates (on the
                                ! unity sphere) of unit vectors
                                ! of the spherical coordinate system
      real      qr_x(nobs)      ! o x-coord of radial       unit vector
      real      qr_y(nobs)      ! o y-coord of radial       unit vector
      real      qr_z(nobs)      ! o z-coord of radial       unit vector
      real      qm_x(nobs)      ! o x-coord of meridional   unit vector
      real      qm_y(nobs)      ! o y-coord of meridional   unit vector
      real      qm_z(nobs)      ! o z-coord of meridional   unit vector
      real      ql_x(nobs)      ! o x-coord of longitudinal unit vector
      real      ql_y(nobs)      ! o y-coord of longitudinal unit vector
                                ! NOTE: ql_z is not needed.


                                ! Interpolation indices/weights:
      integer   ktab(nobs)      ! o vertical   interpolation index
      real      wtab(nobs)      ! o vertical   interpolation weights
      integer   jtab(nobs)      ! o meridional interpolation index
      real      vtab(nobs)      ! o meridional interpolation weights



      integer   nvecs           ! Number of RHS vectors
      integer   nobs_d          ! leading dimension of RHS vector
                                !  as declared in calling program.
                                !  Usually nobs_d = nobs.

      real      b(nobs_d,nvecs) ! Normalized (by innovation stdv)
                                !  RHS vectors. For the convetional
                                !  PSAS analysis system 'b' will
                                !  contain the innovations (O-F).
                                !  However, multiple RHS will be
                                !  necessary for implementation
                                !  analysis error variances by
                                !  randomized trace estimates.
```

OUTPUT PARAMETERS:

```
real       x(nobs_d,nvecs)    ! Solution vectors.
integer    ierr               ! error return code. All is well
                              ! if ierr=0.
```

SEE ALSO:

```
cg_level1()    Pre-conditioner routine.
stdio.h       Include file defining stdandard I/O units
BLAS          Basic linear algebra sub-programs
```

REVISION HISTORY:

```
03apr93   Pfaendtner  Original code
04jun93   Pfaendtner  Modification for dynamic storage on CRAY
07jan94   Sienkiewicz Added pass of trig lat/lon to subroutine
14feb94   da Silva    Fixed search direction bug
09apr94   Pfaendtner  Added prologue
13apr94   Pfaendtner  Added use of libsci routines
03oct94   da Silva    Implemented CRAY specifics with IFDEFs.
04oct94   da Silva    Routine changed name from CONJGR5 to
                      simply CONJGR. Introduced parameter
                      nbandmx.
19Jan95   Guo         Added wobs tables to pass pindx2() values to
                      ??cor1() and ??corx() routines.  One could use
                      rlevs for the same purpose to reduce the over-
                      head, since rlevs has no real purpose in this
                      subroutine and subsequent routines.
02Feb95   Guo         Changed CRAY to _UNICOS for consistency and
                      to follow the guide lines.
11Oct95   Guo         Summary of changes since 02Feb95:
                        + some structural changes for multitasking on
                          C90, including now handling all regions in
                          one cg_level2() call.
                        + modified to accept multi vectors;
                        + replaced multbyC() call to sCxpy() call;
06Feb95   da Silva    Revised prologue, and several minor changes
                      for readability:
                        o name change: from conjgr2() to cg_level2()
                        o removed static allocation IFDEFs;
```

```
                            code now requires Fortran 90 for
                            portability.
                          o simplified main loop
                          o several variable name changes to conform
                            to notation in Golub and van Loan;
                            comments are straight quotation from book.
                          o introduction of f90 assignments whenever
                            possible.
```

SOURCE CODE:

```
      character*9  myname
      parameter   (myname='cg_level2')


!     Local storage (dynamic allocation)
!     ----------------------------------
      include      'mxpass.h'          ! max dimension for sizerr
      real          x_k(nobs,nvecs)    ! solution at kth iteration
      real          r_k(nobs,nvecs)    ! residual at kth iteration
      real          z_k(nobs,nvecs)    ! pre-conditioner at kth iteration
      real          p_k(nobs,nvecs)    ! search direction at kth iteration
      real         Cp_k(nobs,nvecs)    ! Correlation matrix * p_k
      real     r_norm(0:mxpass,nvecs)  ! residual norm

      real    zTr_new(nvecs)           ! z' * r   (new)
      real    zTr_old                  ! z' * r   (old)
                                       ! where z' = transpose(z)

      integer   kt_beg(ktmax,nkr)
      integer   lblkerr(ktmax*nkr)


!     Minor local variables
!     ---------------------
      real      alpha_k, beta, tol
      integer   k, kn, ivec, k_max
      integer   ibeg, ireg, ilen, kt, ikOx, ikFx
      integer   i, ier, lblk


!     Convergence control parameters.
!     -------------------------------
      include     'bands.h'

      include     'stdio.h'

!     Defines kind of covariance matrices
!     -----------------------------------
```

```
        integer    kind_mat
        include    'kind_mats.h'
        include    'kind_covs.h'

!       BLAS functions
!       --------------
        real      sdot, snrm2
        external  sdot, snrm2


!.........................................................................

      call ZEITBEG (cgname(2))

!     Initialization: k=0; x_0=0; r_0=b
!     ---------------------------------
      k = 0
      x_k = 0.
      do ivec=1,nvecs
         r_k(1:nobs,ivec) = b(1:nobs,ivec)
         r_norm(k,ivec) = SNRM2(nobs,r_k(1,ivec),1)
      end do
      kn = 0


!     Iterate...
!     ----------
      k_max = maxpass(2)
      tol = criter(2)
      DO WHILE ( k .le. k_max .and.
     &          (r_norm(kn,1)/r_norm(0,1)) .gt. tol )

         k = k + 1


!        Loop over kt-blocks across regions. The data are sorted by
!        regions, and within each region the obs are sorted by data type
!        (kt). The loop here is over these kt-blocks...
!        ----------------------------------------------------------------
         do lblk = 1, ktmax*nkr

            lblkerr(lblk)=0

            ireg = (lblk-1)/ktmax+1
            kt   = mod(lblk-1,ktmax)+1

            ibeg = kt_beg(kt,ireg)
            ilen = kt_len(kt,ireg)

            ikOx=1
            if((kind_cov.and.kind_covO).ne.0) ikOx=ibeg
            ikFx=1
            if((kind_cov.and.kind_covF).ne.0) ikFx=ibeg
```

```
!           If the kt-block is not empty...
!           -------------------------------
            if ( ilen.gt.0 ) then

!             Invoke the pre-conditioner for each of these
!               univariate kt-blocks
!             -------------------------------------------
              call CG_LEVEL1 ( verbose.and.cgverb(1), kind_cov,
     &                         ireg,  kt, ilen, ks(ikOx),
     &                         nsig_Ou(ikOx), nsig_Oc(ikOx), nsig_F(ikFx),
     &                         qr_x(ibeg), qr_y(ibeg), qr_z(ibeg),
     &                         qm_x(ibeg), qm_y(ibeg), qm_z(ibeg),
     &                         ql_x(ibeg), ql_y(ibeg),
     &                         ktab(ibeg), wtab(ibeg), jtab(ikFx), vtab(ikFx),
     &                         nvecs, nobs,  r_k(ibeg,1),
     &                         z_k(ibeg,1), ier )


!             Error handling. Notice that zeitend() is not balanced,
!             but who cares, since there is a much more serious problem
!             ---------------------------------------------------------
              if(ier.ne.0) then
                if(ier.lt.0) then
                  write(stderr,'(3a,i10)') myname,
     &                         ': insufficient working space in cg_level1(), ',
     &                         'size = ',-ier
                else
                  write(stderr,'(2a,2(a,i3))') myname,
     &                         ': unexpected return from cg_level1(), ',
     &                         'err = ',ier,' with kt = ',kt
                end if
                lblkerr(lblk)=ier
              end if

            end if        ! kt-block is not empty

         end do           ! loop over kt-blocks



!        Additional error handling. This apparently redundant
!        step is only necessary on a parallel enviroment
!        -----------------------------------------------------
         ierr=0
         do lblk=1,ktmax*nkr
            if(lblkerr(lblk).ne.0) then
              ierr=lblkerr(lblk)
              return
            end if
         end do


!        Set search direction, p_k.
!        --------------------------
```

```
         do ivec=1,nvecs

!           if k = 1 { p_1 = z_0 }
!           ----------------------
            if( k.eq.1 ) then

               zTr_new(ivec) = SDOT(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
               call SCOPY(nobs,z_k(1,ivec),1,p_k(1,ivec),1)


!           else { beta_k = r_{k-1}^T z_{k-1} / r_{k-1}^T z_{k-2}
!                   p_k = z_k-1 + \beta_k p_{k-1} }
!           -----------------------------------------------------
            else

               zTr_old = zTr_new(ivec)
               zTr_new(ivec) = SDOT(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
               beta    = zTr_new(ivec) / zTr_old
               call SAXPY(nobs,beta,p_k(1,ivec),1,z_k(1,ivec),1)
               call SCOPY(nobs,z_k(1,ivec),1,p_k(1,ivec),1)

            end if

         end do        ! loop over RHS vectors


!        q_k = C p_k
!        -----------
         kind_mat=kind_Rmat
         call sCxpy ( kind_mat, kind_cov,
     &                nkr, kr_beg, kr_len, kt_len,
     &                nobs, ks,nsig_Ou, nsig_Oc, nsig_F,
     &                qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,
     &                ktab, wtab, jtab, vtab,
     &                nvecs, nobs, p_k, nobs, Cp_k,
     &                ierr )


!        Error handling
!        --------------
         if ( ierr .ne. 0 ) then
              if(ierr.lt.0) then
                write(stderr,'(3a,i10)') myname,
     &                       ': insufficient working space in sCxpy(), ',
     &                       'size = ',-ierr
              else
                write(stderr,'(3a,i3)') myname,
     &                       ': unexpected return from sCxpy(), ',
     &                       'err = ',ierr
              end if
              call PSASexit(2,myname)
         end if
```

```
!        For each RHS vector
!        -------------------
         do ivec=1,nvecs

!           alpha_k = z_{k-1}^T r_{k-1} / p_k^T q_k
!           ---------------------------------------
            alpha_k = zTr_new(ivec) /
     &                SDOT(nobs,p_k(1,ivec),1,Cp_k(1,ivec),1)

!           x_k = x_{k-1} + alpha_k p_k
!           ---------------------------
            call SAXPY(nobs,+alpha_k, p_k(1,ivec),1,x_k(1,ivec),1)

!           r_k = r_{k-1} - alpha_k q_k
!           ---------------------------
            call SAXPY(nobs,-alpha_k,Cp_k(1,ivec),1,r_k(1,ivec),1)

         end do


!        Residual norm at end of this iteration
!        --------------------------------------
         kn = kn + 1
         if ( kn .gt. MXPASS ) kn = 1   ! cyclic storage
         do ivec=1,nvecs
            r_norm(kn,ivec) = SNRM2(nobs,r_k(1,ivec),1)
         end do


      END DO  ! end of CG iteration


!     Convergence achieved
!     --------------------
      if( (r_norm(kn,1)/r_norm(0,1)) .le. tol .and. verbose ) then
           write(stdout,'(2a)') myname,': convergence achieved'


!     Maximum number of iterations exceeded
!     -------------------------------------
      else if ( verbose ) then
           write(stdout,'(2a)') myname,
     &                  ': maximum number of iterations exceeded'

      end if


!     Prints summary
!     --------------
      if ( verbose ) then
           call CGNORM ( myname, criter(2), mxpass, k, nvecs, r_norm, nobs )
      end if
```

```
!     Return kth iterate as solution
!     ------------------------------
      do ivec=1,nvecs
          x(1:nobs,ivec) = x_k(1:nobs,ivec)
      end do


!     All done
!     --------
      call ZEITEND

      return
      end
```

A.7 cg_level1()

Solves the linear system of equations

$$\hat{C}x = b$$

where $\hat{C}$ is a simplified version of innovation covariance matrix, and b is a set of multiple right-hand-sides. The matrix $\hat{C}$ consists of regional diagonal blocks of the the correlation matrix C. This routine is meant to be a pre-conditioner for routine `cg_level2()`. When perfoming a global analysis with PSAS, the RHS is simply the innovation (O-F) normalized by its standard deviation. The multiple RHS are necessary for the estimate of analysis error variances by means of randomized trace estimates.

The *Pre-conditioned Conjugate Gradient* algorithm is standard and closely follows

Golub, G. H. and C. F. van Loan, 1989: *Matrix Computations*, 2nd Edition, The John Hopkins University Press, 642pp.

and is reproduced in the prologue of routine `cg_main()`. The pre-conditioner for this routine is implemented using LAPACK's Cholesky solver [routines spptrf() and spptrs()]. This pre-conditoner solves a much smaller problem, considering only diagonal blocks of C with a "couple" of profiles.

CALLING SEQUENCE:

```
      call CG_LEVEL1 ( verbose, kind_cov,
     &                 ireg, kt,
     &                 nobs, ks, nsig_Ou, nsig_Oc, nsig_F,
     &                 qr_x, qr_y, qr_z, qm_x, qm_y, qm_z, ql_x, ql_y,
     &                 ktab, wtab, jtab, vtab,
     &                 nvecs, nobs_d, b, x, ierr  )
```

INPUT PARAMETERS:

```
      implicit  NONE

      logical   verbose            ! if .true. prints out all kind
                                   !  of informational output to stdout.

      integer   kind_cov           ! specifies the kind of covariance
                                   ! matrix

      integer   ireg               ! PSAS region index
      integer   kt                 ! GEOS/DAS data-type index
```

```
integer   nobs               ! number of observations
integer   ks(nobs)           ! sounding index

                             ! Observation/forecast errors stdv
                             ! normalized by innovation (O-F) stdv:
real      nsig_Ou(nobs)      ! o normalized spatially uncorrelated
                             !   observation error stdv
real      nsig_Oc(nobs)      ! o normalized spatially correlated
                             !   observation error stdv
real      nsig_F(nobs)       ! o normalized forecast error stdv


                             ! Cartesian coordinates (on the
                             ! unity sphere) of unit vectors
                             ! of the spherical coordinate system
real      qr_x(nobs)         ! o x-coord of radial       unit vector
real      qr_y(nobs)         ! o y-coord of radial       unit vector
real      qr_z(nobs)         ! o z-coord of radial       unit vector
real      qm_x(nobs)         ! o x-coord of meridional   unit vector
real      qm_y(nobs)         ! o y-coord of meridional   unit vector
real      qm_z(nobs)         ! o z-coord of meridional   unit vector
real      ql_x(nobs)         ! o x-coord of longitudinal unit vector
real      ql_y(nobs)         ! o y-coord of longitudinal unit vector
                             ! NOTE: ql_z is not needed.


                             ! Interpolation indices/weights:
integer   ktab(nobs)         ! o vertical   interpolation index
real      wtab(nobs)         ! o vertical   interpolation weights
integer   jtab(nobs)         ! o meridional interpolation index
real      vtab(nobs)         ! o meridional interpolation weights


integer   nvecs              ! Number of RHS vectors
integer   nobs_d             ! leading dimension of RHS vector
                             !  as declared in calling program.
                             !  Usually nobs_d = nobs.

real      b(nobs_d,nvecs)    ! Normalized (by innovation stdv)
                             !  RHS vectors. For the convetional
                             !  PSAS analysis system 'b' will
                             !  contain the innovations (O-F).
                             !  However, multiple RHS will be
                             !  necessary for implementation
                             !  analysis error variances by
                             !  randomized trace estimates.
```

OUTPUT PARAMETERS:

```
real       x(nobs_d,nvecs)     ! Solution vectors.
integer    ierr                ! error return code. All is well
                               ! if ierr=0.
```

SEE ALSO:

```
stdio.h      Include file defining stdandard I/O units
LAPACK       Linear Algebra PACKage
BLAS         Basic linear algebra sub-programs
```

REVISION HISTORY:

```
03apr93   Pfaendtner  Original code
04jun93   Searl       Modification for dynamic storage on CRAY
07jan94   Sienkiewicz Added pass of trig lat/lon to subroutine
14feb94   da Silva    Fixed search direction bug
09apr94   Pfaendtner  Added prologue
13apr94   Pfaendtner  Added use of libsci routines
03oct94   da Silva    Implemented CRAY specifics with IFDEFs.
04oct94   da Silva    Routine changed name from CONJGR5 to
                      simply CONJGR. Introduced parameter
                      nbandmx.
19Jan95   Guo         Added wobs tables to pass pindx2() values to
                      ??cor1() and ??corx() routines.  One could use
                      rlevs for the same purpose to reduce the over-
                      head, since rlevs has no real purpose in this
                      subroutine and subsequent routines.
02Feb95   Guo         Changed CRAY to _UNICOS for consistency and
                      to follow the guide lines.
11Oct95   Guo         Summary of changes since 02Feb95:
                        + some structural changes for multitasking on
                          C90, including now handling all regions in
                          one cg_level2() call.
                        + modified to accept multi vectors;
                        + replaced multbyC() call to sCxpy() call;
06Feb95   da Silva    Revised prologue, and several minor changes
                      for readability:
                        o name change: from conjgr1() to cg_level1()
                        o removed static allocation IFDEFs;
```

```
                                code now requires Fortran 90 for
                                portability.
                              o simplified main loop
                              o several variable name changes to conform
                                to notation in Golub and van Loan;
                                comments are straight quotation from book.
                              o introduction of f90 assignments whenever
                                possible.
```

SOURCE CODE:

```
      character*9  myname
      parameter   (myname='cg_level1')


!     Local storage (dynamic allocation)
!     ----------------------------------
      include          'mxpass.h'

      real       corr(nobs*(nobs+1)/2)     ! Temporary correlation matrix
      real       corrM(nobs*(nobs+1)/2)    ! Innovation correlation  matrix
      real       corrI(nobs*(nobs+1)/2)    ! Inverse of corrM
      real          x_k(nobs,nvecs)        ! solution at kth iteration
      real          r_k(nobs,nvecs)        ! residual at kth iteration
      real          z_k(nobs,nvecs)        ! pre-conditioner at kth iteration
      real          p_k(nobs,nvecs)        ! search direction at kth iteration
      real         Cp_k(nobs,nvecs)        ! Correlation matrix * p_k
      real      r_norm(0:mxpass,nvecs)     ! residual norm

      real     zTr_new(nvecs)              ! z' * r   (new)
      real     zTr_old                     ! z' * r   (old)


!     Minor local storage (static allocation)
!     ---------------------------------------
      integer      ivec
      integer      k_max
      real         tol
      integer      begin_blk, next_blk, begin_sav
      real         endqrx
      logical      next

      character*1  Mtyp
      integer      ij

      real         alpha_k, beta
      integer      N_diverg
      integer      k, m, i, j, kn, km
      logical      converging
```

```
      logical      solved
      integer      nshift
      integer      mshift
      parameter   (mshift=10)
      real         dshift
      parameter   (dshift=.1/mshift)


      include     'bands.h'         ! Convergence control parameters

      include     'stdio.h'         ! standard I/O

      include     'realvals.h'      ! machep look-alike

      include     'kind_covs.h'     ! kind of covariance matrices


      logical     setCorF
      parameter  (setCorF=.true.)


!     BLAS functions
!     --------------
      real        sdot, snrm2
      external    sdot, snrm2

      integer     lnblnk, luavail
      external    lnblnk, luavail

!.........................................................................

      call ZEITBEG (cgname(1))


!     Initialization: k=0; x_0=0; r_0=b
!     ---------------------------------
      k = 0
      x_k = 0.
      do ivec=1,nvecs
         r_k(1:nobs,ivec) = b(1:nobs,ivec)
         r_norm(k,ivec) = SNRM2(nobs,r_k(1,ivec),1)
      end do
      kn = 0


!     Compute the block matrix corrM to work on
!     -----------------------------------------
       corr = 0.
      corrI = 0.
      corrM = 0.
      call CG_BLOCKS()            ! this an internal routine
```

```
!     Iterate...
!     ----------
      k_max = maxpass(1)
      tol = criter(1)
      N_diverg = 0
      converging = .true.
      DO WHILE ( converging .and.
     &           k .le. k_max .and.
     &           (r_norm(kn,1)/r_norm(0,1)) .gt. tol )

         k = k + 1

!        Preconditioner for level 1 (one region, one kt) is direct
!        solver on diagonal sub-blocks. It makes sure that
!        soundings are kept together (group by qr_x)
!        ---------------------------------------------------------
         call CG_LEVEL0()         ! this an internal routine


!        if k = 1 { p_1 = z_0 }
!        ----------------------
         if( k.eq.1 ) then
            do ivec=1,nvecs
               zTr_new(ivec) = SDOT(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
               call SCOPY(nobs,z_k(1,ivec),1,p_k(1,ivec),1)
            end do


!        else { beta_k = r_{k-1}^T z_{k-1} / r_{k-1}^T z_{k-2}
!               p_k = z_k-1 + \beta_k p_{k-1} }
!        -----------------------------------------------------
         else
            do ivec=1,nvecs
               zTr_old = zTr_new(ivec)
               zTr_new(ivec) = SDOT(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
               beta = zTr_new(ivec) / zTr_old
               call SAXPY(nobs,beta,p_k(1,ivec),1,z_k(1,ivec),1)
               call SCOPY(nobs,z_k(1,ivec),1,p_k(1,ivec),1)
            end do
         end if


!        For each RHS vector
!        -------------------
         do ivec=1,nvecs

!           q_k = C p_k
!           -----------
            call SSPMV('U',nobs, 1.,corrM,p_k(1,ivec),1,
     &                 0.,  Cp_k(1,ivec),1)


!           alpha_k = z_{k-1}^T r_{k-1} / p_k^T q_k
!           ---------------------------------------
```

```
          alpha_k = zTr_new(ivec) /
     &              SDOT(nobs,p_k(1,ivec),1,Cp_k(1,ivec),1)

!         x_k = x_{k-1} + alpha_k p_k
!         ---------------------------
          call SAXPY(nobs,+alpha_k, p_k(1,ivec),1,x_k(1,ivec),1)

!         r_k = r_{k-1} - alpha_k q_k
!         ---------------------------
          call SAXPY(nobs,-alpha_k,Cp_k(1,ivec),1,r_k(1,ivec),1)

       end do


!      Residual norm at end of this iteration
!      --------------------------------------
       km = kn
       kn = kn + 1
       if ( kn .gt. MXPASS ) kn = 1    ! cyclic storage
       do ivec=1,nvecs
          r_norm(kn,ivec) = SNRM2(nobs,r_k(1,ivec),1)
       end do


!      Detect divergence: one iteration is termed "divergent" if the
!       residual increases instead of decreasing. N_diverg
!       records how many times this happens
!      -----------------------------------------------------------
       if ( r_norm(kn,ivec) .ge. r_norm(km,ivec) ) then
            N_diverg = N_diverg + 1
       end if


!      The CG process is called "divergent" if the number
!       of divergent iterations exceeds a pre-determined
!       number (minmax(1))
!      -------------------------------------------------
       converging = N_diverg .lt. minmax(1)


    END DO  ! end of CG iteration



!   Convergence achieved
!   --------------------
    if( (r_norm(kn,1)/r_norm(0,1)) .le. tol .and. verbose ) then
         write(stdout,'(2a)') myname,': convergence achieved'


!   Divergence detected
!   -------------------
    else if ( .not. converging .and. verbose ) then
         write(stdout,'(2a)') myname,
```

```
     &                   ': conjugate gradient is not converging. '

!     Maximum number of iterations exceeded
!     -------------------------------------
      else if ( verbose ) then
           write(stdout,'(2a)') myname,
     &                   ': maximum number of iterations exceeded'

      end if

!     Prints summary
!     --------------
      if ( verbose ) then
           call CGNORM ( myname, criter(1), mxpass, k, nvecs, r_norm, nobs )
      end if

!     Return kth iterate as solution
!     ------------------------------
      do ivec=1,nvecs
          x(1:nobs,ivec) = x_k(1:nobs,ivec)
      end do

!     All done
!     --------
      call ZEITEND

      return


      CONTAINS
!     --------
```

A.8 cg_blocks()

Computes innovation correlation blocks. This is an internal routine of CG_LEVEL1().

CALLING SEQUENCE:

```
call cg_blocks()
```

INPUT PARAMETERS:

```
none.
```

OUTPUT PARAMETERS:

```
None explicitly, but corrM is calculated here.
```

SEE ALSO:

```
cg_level1()   parent routine.
```

REVISION HISTORY:

```
06Feb96   da Silva   Moved from body of CG_LEVEL1 for redability.
```

SOURCE CODE:

```
      Mtyp='Z'
      if ((kind_cov.and.kind_covO).ne.0) then


!         Construct spatially correlated observation error correlation
!         matrix
!         ------------------------------------------------------------
          call DiagCorO ( kt,nobs,ks,qr_x,qr_y,qr_z,ktab,wtab,
     &                    Mtyp,corr,ierr)

!         Error handling
!         --------------
          if(ierr.ne.0) then
            write(stderr,'(a,2(a,i3))') myname,
     &                   ': unexpected variable type for diagcorO(), kt = ',kt,
     &                   ', ierr =',ierr
            return
          end if


          If ( Mtyp.eq.'U' .or. Mtyp.eq.'u' ) then
            do j=1,nobs
              ij=j*(j-1)/2
              do i=1,j
                corrM(ij+i)=nsig_Oc(i)*corr(ij+i)*nsig_Oc(j) + corrM(ij+i)
              end do
            end do
          else if ( Mtyp .eq. 'I' .or. Mtyp .eq. 'i' ) then
            do j=1,nobs
              ij=j*(j+1)/2
              corrM(ij)=nsig_Oc(j)*nsig_Oc(j) + corrM(ij)
            end do
          end if


!         Construct uncorrelated observation error correlation
!         ----------------------------------------------------
          call DiagCorU (kt,nobs,ks,ktab,wtab,Mtyp,corr,ierr)

!         Error handling
!         --------------
          if(ierr.ne.0) then
            write(stderr,'(a,2(a,i3))') myname,
     &              ': unexpected variable type for diagcorU(), kt = ',kt,
     &              ', ierr =',ierr
            return
          end if
```

```
          If(Mtyp.eq.'U'.or.Mtyp.eq.'u') then
            do j=1,nobs
              ij=j*(j-1)/2
              do i=1,j
                corrM(ij+i)=nsig_Ou(i)*corr(ij+i)*nsig_Ou(j) + corrM(ij+i)
              end do
            end do
          elseif(Mtyp.eq.'I'.or.Mtyp.eq.'i') then
            do j=1,nobs
              ij=j*(j+1)/2
              corrM(ij)=nsig_Ou(j)*nsig_Ou(j) + corrM(ij)
            end do
          end if

        end if

        Mtyp='Z'
        if ((kind_cov.and.kind_covF).ne.0) then


        call DiagCorF ( kt,nobs,qr_x,qr_y,qr_z,qm_x,qm_y,qm_z,
     &                  ql_x,ql_y,ktab,wtab,
     &                  Mtyp,corr,ierr)

!       Error handling
!       --------------
        if(ierr.ne.0.or.Mtyp.eq.'E') then
            write(stderr,'(a,2(a,i3))') myname,
     &               ': unexpected variable type for diagcorF(), kt = ',kt,
     &               ', ierr =',ierr
            return
        end if


        if(Mtyp.eq.'U'.or.Mtyp.eq.'u') then
           do j=1,nobs
              ij=j*(j-1)/2
              do i=1,j
                 corrM(ij+i)=nsig_F(i)*corr(ij+i)*nsig_F(j) + corrM(ij+i)
              end do
           end do
        end if
      end if


!     Al done
!     -------
      return

      end subroutine CG_BLOCKS
```

A.9 cg_level0()

Implements the pre-conditioner for cg_level1(). The pre-conditioner for level 1 (one region, one kt) is direct solver on diagonal sub-blocks. It makes sure that soundings are kept together (group by qr_x). This is an internal routine of `CG_LEVEL1()`.

CALLING SEQUENCE:

```
call cg_level0()
```

INPUT PARAMETERS:

```
none.
```

OUTPUT PARAMETERS:

```
None explicitly, but z_k is calculated here.
```

SEE ALSO:

```
cg_level1()   parent routine.
```

REVISION HISTORY:

```
06Feb96   da Silva   Moved from body of CG_LEVEL1 for redability.
```

SOURCE CODE:

```
!     Make a copy of the current residual
!     ----------------------------------
      do ivec=1,nvecs
         call SCOPY(nobs,r_k(1,ivec),1,z_k(1,ivec),1)
      end do

      begin_blk = 1
      begin_sav = 1
      DO WHILE ( begin_blk .le. nobs )

!        It (next_blk) is actually the end-of-this-block
!        ----------------------------------------------
         next_blk = min(begin_blk+msmall-1,nobs)
         endqrx = qr_x(next_blk)

!           Search for end of this sounding (at end of msmall sized
!           block) and set block break where soundings change
!           Tests are made in sequence to avoid qr_x(nobs+1) ever
!           being referenced.
!           ---------------------------------------------------------
         next=.true.
         do while ( next )
            next_blk = next_blk + 1

            next=next_blk.le.nobs
            if(next) next=qr_x(next_blk).eq.endqrx
         end do

         m = next_blk - begin_blk

         if(k.eq.1) then

            nshift=0
            solved=.false.
            call smex(corrM,nobs,begin_blk,m,corrI(begin_sav))

            do while(.not.solved)
               call SPPTRF('U',m,corrI(begin_sav),ierr)

               if( ierr.ne.0 ) then

                  write(stdout,'(a,5(a,i3),a,i5)') myname,
     &                 ': SPPTRF() error ',ierr,
     &                 ': nshift=',nshift,
     &                 ' region=',ireg,
     &                 ' type=',kt,
     &                 ' msmall=',m,
     &                 ' begblk=',begin_blk
```

```
            nshift=nshift+1
            if(nshift.gt.mshift) then
               write(stderr,'(a,2(a,i4),a)') myname,
     &               ': err = ',ierr,' in SPPTRF() after ',nshift,
     &               ' tries'
               return
            end if

            call smex(corrM,nobs,begin_blk,m,corrI(begin_sav))
            call smexsh(corrI(begin_sav),m,nshift*dshift)

         else
            solved=.true.
         end if              ! error
       end do                ! .not.solved
     end if                  ! k.eq.1

     call SPPTRS('U',m,nvecs,corrI(begin_sav),z_k(begin_blk,1),
     &              nobs,ierr)

     if(ierr.ne.0) then      ! if it ever happens.
        write(stderr,'(a,2(a,i2))') myname,
     &        ': err = ',ierr,' from SPPTRS() with m = ',m,
     &        ' and begin_blk = ',begin_blk
        return
     end if

     begin_blk = next_blk
     begin_sav = begin_sav+m*(m+1)/2

   end do                    ! next block (starting from begin_blk)?

!  All done
!  --------
   return
   end subroutine CG_LEVEL0

!.........................................................................

   end subroutine CG_LEVEL1
```

References

Anderson, E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, 1992: LAPACK User's Guide. *Society for Industrial and Applied Mathematics*, Philadelphia, PA, 235pp.

Cohn, S. E., 1991: New observation processing method. Manuscript, unpublished notes.

Courtier, P., E. Andersson, W. Heckley, G. Kelly, J. Pailleux, F. Rabier, J.-N. Thepaut, P. Unden, D. Vasiljevic, C. Cardinali, J. Eyre, M. Hamrud, J. Haseler, A. Hollingsworth, A. Mc Nally, and A. Stoffelen, 1993: Variational Assimilation at ECMWF. *ECMWF Technical Memorandum*, No. 194. Reading, England, 84pp.

Daley, R., 1991: *Atmospheric Data Analysis.* Cambridge University Press. New York, 457pp. ISBN 0-521-38215-7.

Gaspari, G. and S. E. Cohn, 1996: Construction of Correlation Functions in Two and Three Dimensions. *DAO Office Note 96-03.* Data Assimilation Oddice, Code 910.3, Goddard Space Flight Center, Greenbelt, MD 20771.

Golub, G. H. and C. F. van Loan, 1989: *Matrix Computations*, 2nd Edition, The Johns Hopkins University Press, 642pp.

Guo, J. and A. da Silva, 1995: Computational aspects of Goddard's Physical-space Statistical Analysis System (PSAS). *Second UNAM-Cray Supercomputing Conference.* Mexico City, Mexico, June 1995.

Parrish, D.F. and J.C. Derber, 1992: The National Meteorlogical Center's statistical spectral interpolation analysis system. *Mon. Wea. Rev.*, **109**, 1747-1763.

Pfaendtner, J., 1996: Notes on the Icosahedral Domain Decomposition in PSAS. *DAO Office Note 96-04.* Data Assimilation Oddice, Code 910.3, Goddard Space Flight Center, Greenbelt, MD 20771.

Pfaendtner, J., S. Bloom, D. Lamich, M. Seablom, M. Sienkiewicz, J. Stobie, A. da Silva, 1995: Documentation of the Goddard Earth Observing System (GEOS) Data Assimilation System–Version 1. *NASA Tech. Memo. No. 104606*, Vol. 4, Goddard Space Flight Center, Greenbelt, MD 20771. Available electronically on the World Wide Web as `ftp://dao.gsfc.nasa.gov/pub/tech_memos/volume_4.ps.Z`

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, 1992: *Numerical recipes in Fortran*, Second Ed. Cambridge University Press, New York, USA, 963pp.

Schubert, S.D., R. B. Rood, and J. Pfaendtner, 1993: An assimilated data set for earth science applications. *Bul. Amer. Meteor. Soc.*, **74**, 2331-2342.

Schubert, S., C.-K. Park, Chung-Yu Wu, W. Higgins, Y. Kondratyeva, A. Molod, L. Tkacs, M. Seablom, and R. Rood, 1995a: A Multiyear Assimilation with the GEOS-1 System: Overview and Results. *NASA Tech. Memo. 104606*, Vol. **6.** Goddard Space Flight Center, Greenbelt, MD 20771. Available electronically on the World Wide Web as `ftp://dao.gsfc.nasa.gov/pub/tech_memos/volume_6.ps.Z`

Schubert, S. D. and R. Rood, 1995b: Proceedings of the Workshop on the GEOS-1 Five-Year Assimilation. *NASA Tech. Memo. 104606*, Vol. **7**, Goddard Space Flight Center, Greenbelt, MD 20771. Available electronically on the World Wide Web as `ftp://dao.gsfc.nasa.gov/pub/tech_memos/volume_7.ps.Z` da Silva, A. and C. Redder, 1995: Documentation of the GEOS/DAS Observation Data Stream (ODS) Version 1.0. *DAO Office Note 95-01.*. Data Assimilation Office, Goddard Space Flight Center, Greenbelt, MD 20771.

da Silva, A., J. Pfaendtner, J. Guo, M. Sienkiewicz and S. E. Cohn, 1995: Assessing the Effects of Data Selection with DAO's Physical-space Statistical Analysis System. *International Symposium on Assimilation of Observations*, Tokyo, Japan, 133-17 March 1995, 273–278.